SPRITE GRAPHICS

FOR THE

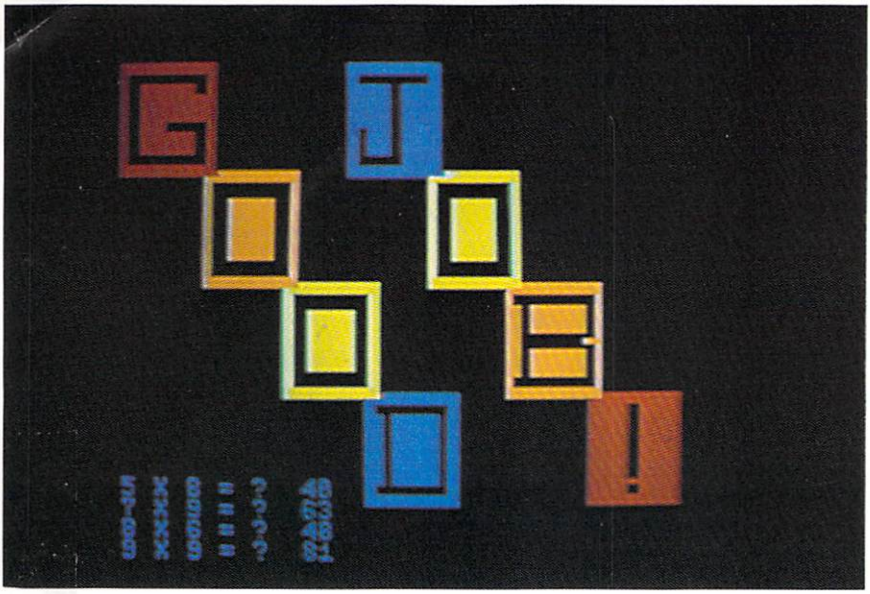
COMMODORE 64™



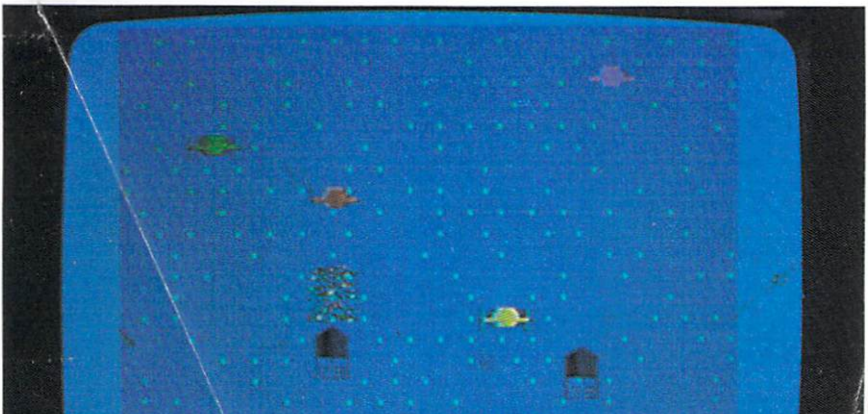
INCLUDES 25 READY-TO-RUN PROGRAMS

SALLY GREENWOOD LARSEN

A SPECTRUM BOOK
PRENTICE-HALL/MICRO TEXT

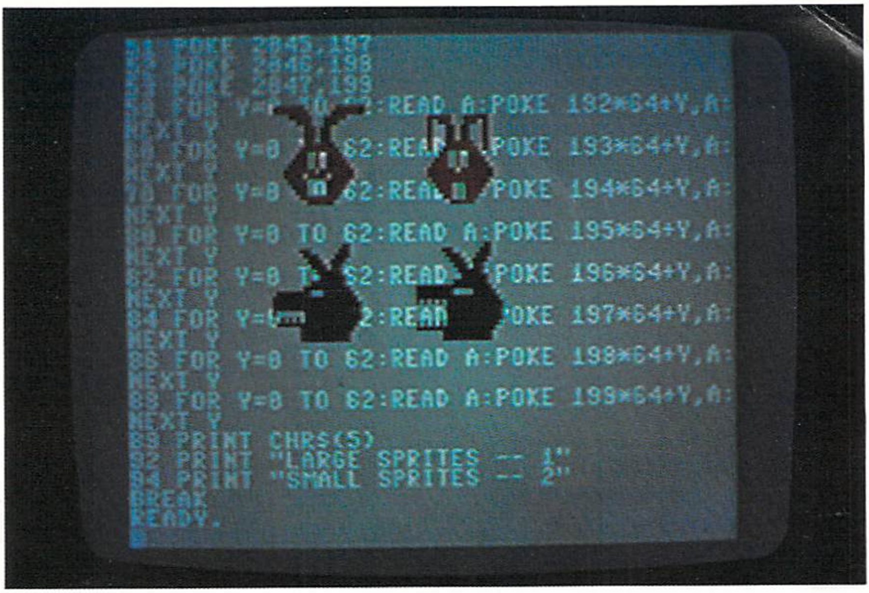


Sprite routines make excellent additions to liven up educational programs.

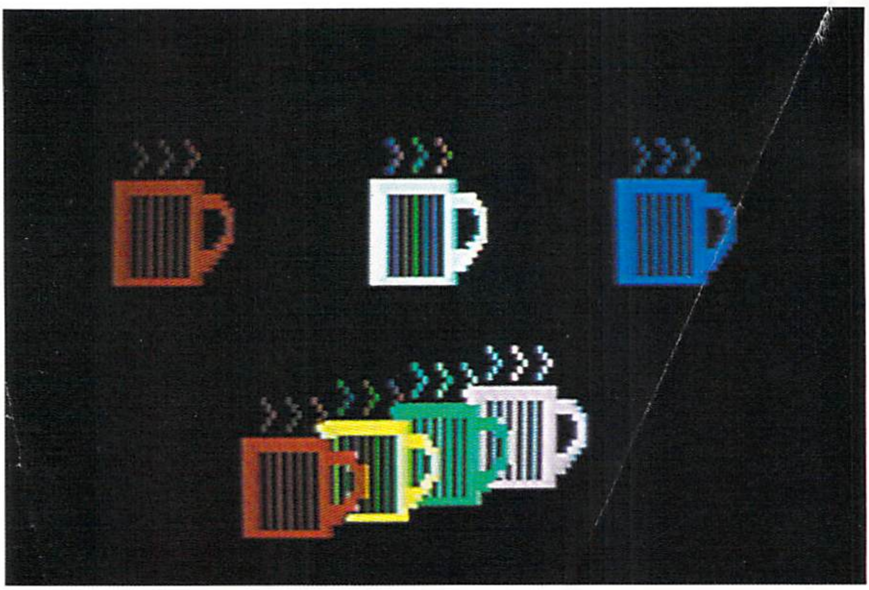


Built-in collision detection makes Sprite graphics a natural for action games.

photo credit Greg Willmore

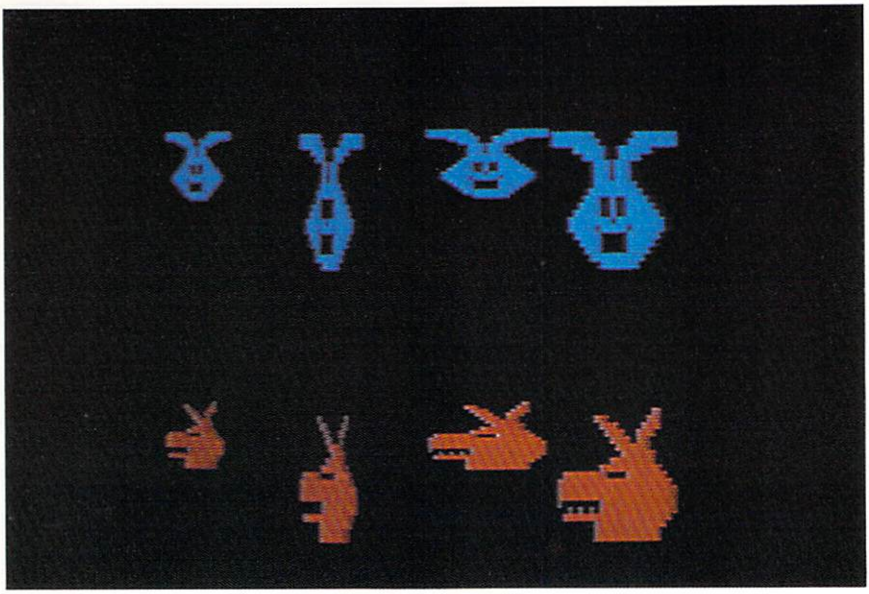


Sprites can be animated by rapidly displaying slightly different versions of the same figure.

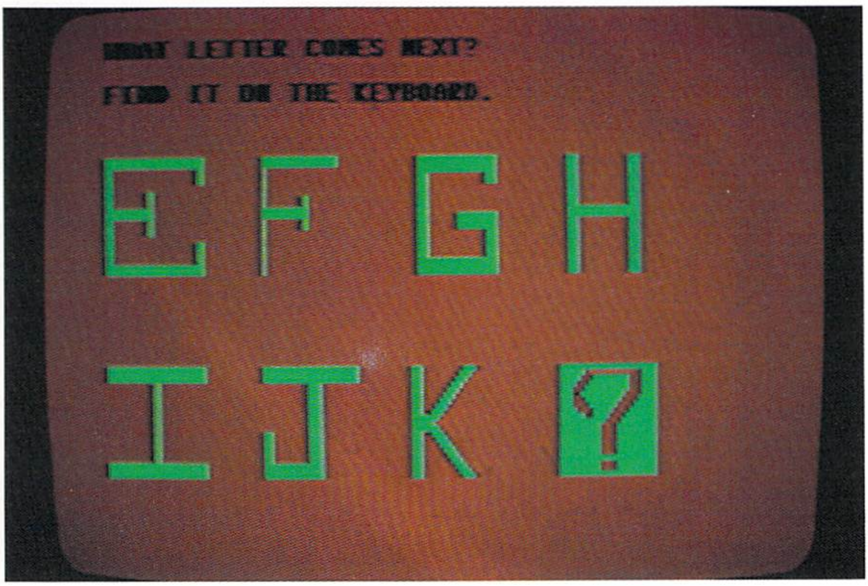


Sprite priorities are automatic, making three-dimensional graphics effects simple.

photo credit Greg Willmore

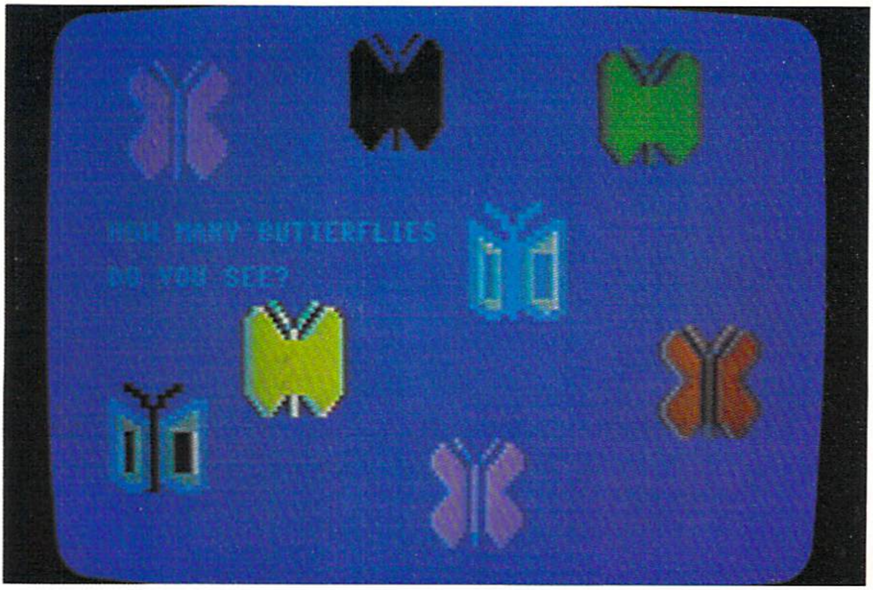


Automatic Sprite expansion along either axis gives three views for the price of one.

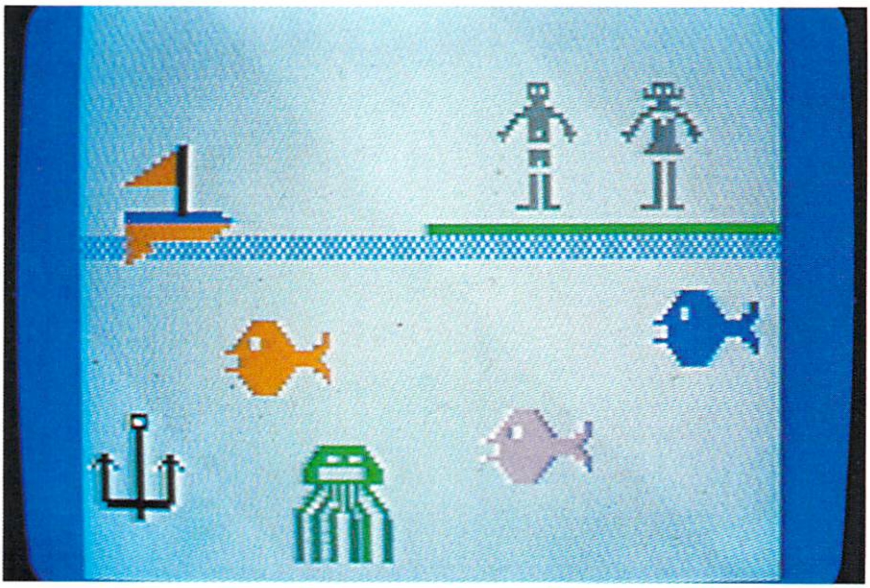


Switching Sprite memory pointers lets us instantly reference any of these alphabet shapes.

photo credit Greg Willmore



Single and multi-color butterfly Sprites help preschoolers learn to count.



Sprites can easily be combined with any other type of Commodore 64 graphics.

photo credit Greg Willmore

SPRITE GRAPHICS FOR THE COMMODORE 64™

SALLY GREENWOOD LARSEN

A SPECTRUM BOOK
Micro Text Publications, Inc.
Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data

Larsen, Sally Greenwood.
Sprite graphics for the Commodore 64.

"A Spectrum Book."
Includes index.
1. Computer graphics. 2. Commodore 64 (Computer)
I. Title.
T385.L37 1983 001.64'43 83-21164
ISBN 0-13-838144-5
ISBN 0-13-838136-4 (pbk.)

A Spectrum Book. Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-838144-5

ISBN 0-13-838136-4 {PBK.}

This book is available at a special discount when ordered in bulk quantities. Contact Prentice-Hall, Inc., General Publishing Division, Special Sales, Englewood Cliffs, N.J. 07632.

Prentice-Hall International, Inc., *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Prentice-Hall of Southeast Asia Pte. Ltd., *Singapore*
Whitehall Books Limited, *Wellington, New Zealand*
Editora Prentice-Hall do Brasil Ltda., *Rio de Janeiro*

DEDICATION

To the original members of the Old Fashioned Recipe quartet:

Barb Gluth
Ronnie McGinn
Geri Shelton

whose music, humor, and caring gave me a year's worth of wonderful memories. May our friendships last for the rest of our lives, no matter how many miles may separate us.

ACKNOWLEDGMENTS

Putting together a book is always a major undertaking. I am happy to be able to give credit to those people who made it all possible.

Alan Rose, my publisher at Micro Text Publications, was the driving force behind this work. Without him, you would not be reading this book.

Greg Willmore, of Willmore Graphics in Milan, Illinois, produced the color photographs. He spent extra time on this project, looking for just the right color combinations and camera settings to arrive at the excellent results he was able to obtain. Thanks, Greg. You did a first-class job.

Whenever one deals with computers, there will always be unforeseen difficulties with hardware and software. (At this point, I can think of several new corollaries for Murphy's Law.) But I was fortunate enough to have the help and support of Cosmos Computers, Bettendorf, Iowa. The owner, John Yigas, and his staff went out of their way to be there with whatever I needed, whenever I needed it. Thank you to John for the hardware help, to Bob Suominen for the printer advice, to Elaine Snyder for the tips on selecting word processing software, and to Jay Martin, programmer extraordinaire, who guided me through the dark forest of Boolean expressions. I appreciate the help all of you gave to me so willingly.

TABLE OF CONTENTS

INTRODUCTION

May I make an observation here?

I'd be willing to bet that you've already read the chapter in the *Commodore 64 User's Guide* which pertains to Sprite graphics. I'd also be willing to bet that you understood very little of it the first time through, right?

Right.

The whiz kids next door have been conquering the universe for a month now on their home computer, but all you've been able to do is type in the sample Sprite program and watch the three balloons float around on the screen. And to be truthful, you aren't even sure why that program works. Right?

Right.

Then this is the book for you. I was in the same position you are, not so long ago, and I have carefully explained in this book everything I wish I'd known when I started with Sprite graphics. They really are as powerful, as impressive, and as easy to use as the advertisements proclaim — once you understand how and why they work.

This book was written with the beginning or intermediate programmer in mind. You do need a general understanding of how your computer operates, and how to write simple programs in BASIC, but I will supply the rest. I've even included a chapter on binary notation and arithmetic, if you need a review (or you're learning it for the first time).

I can't promise that you'll be conquering the universe within a half hour, but I can guarantee that when the kids next door ask how you're getting along with your computer, you'll be able to show them some animated graphics that will knock their computerized socks off.

CHAPTER 1

OVERVIEW OF SPRITE GRAPHICS

Many of us would like to use high resolution graphics in our programs, but find them to be too much work. Not only do you have to plot each little dot in your figure, and go through BASIC gymnastics to move it around on the screen as a unit, but all too often the results are disappointing. The movement is too slow or too jerky for your purposes, and you end up tossing out the whole program.

Sprite graphics, a specially designed system of high resolution color graphics built into the Commodore 64 computer, solves many of these problems. It allows both experienced and beginning programmers to produce professional-looking animated graphics with a minimum of coding.

WHAT IS A SPRITE?

Each 24 by 21 dot Sprite is a shape designed by the programmer. It can be all one color, or a combination of up to four colors. The Sprite is placed on the screen by defining the X and Y coordinates of the location desired. Movement is achieved by "stepping" the Sprite through the coordinate values.

One of the most powerful (yet easiest to use) features of Sprite graphics is dominance — choosing which Sprite should have the "right of way" when they pass each other on the screen. Up to eight Sprites can be displayed on the screen at once, each moving in whatever direction the programmer chooses. Priority is defined by the numbering system for the Sprites. Sprites with lower numbers will have priority over higher-numbered Sprites. So, if Sprites #1 and #3 meet, Sprite #1 will pass "in front of" Sprite #3. This three-dimensional effect is further enhanced when the colors of Sprite #3 appear through any "holes" in Sprite #1. This gives a marvelous feeling of depth to the animation possible with Sprite graphics.

A built-in collision detection system allows the programmer to direct the action based on Sprites coming into contact with one another (a natural for designing arcade-style games). Collisions can also be detected between Sprites and any background figures, such as the walls of a maze, or a boundary, or even a word printed on the screen.

Sprites can be made larger than the originally defined size by a very simple process. A Sprite can be expanded along the X-axis, along the Y-axis, and along both axes.

Some Basic Sprite Facts

Before we look at a sample program, let's define a few basic terms and ideas concerning Sprites.

The Commodore 64 is able to use Sprite graphics because it has inside it a special microprocessor chip, referred to as the Video Chip. This chip contains 46 areas to store information, called REGISTERS. Each register is one byte long. That byte is further broken down into eight BITS of information. Sometimes we will be using the decimal value of the number stored in each register. Other times it will be necessary to examine individual bits within the register. If you don't know a bit from a byte, and a binary number sounds faintly unsavory to you, you probably should read Chapter 13 before you go much further. It contains a review of binary notation and how bits and bytes work together. A good working knowledge of binary notation is a must for programming with Sprites.

We must have a way to reference each of the registers in the video chip. Each has a number, but these numbers are rather large. (They start at 53248 and get worse.) Since each of the registers we are examining is somewhere in the video chip, and we know that the beginning address of the video chip is 53248, we can take advantage of that fact and express each register as an offset from the beginning of the video chip. Instead of referring to the Sprite Enable Register as location 53264, we can call it V+16, where V=53248. Either expression has the same value, but it is much easier for the memory to deal with V+16.

Now you are wondering whose memory I mean — yours or the computer's? Actually, both! The advantage for the memory between your ears is obvious. For the computer, the expression V+16 takes less space in its memory; and is therefore more efficient.

The Case For Clarity

Speaking of efficiency, I can just hear the comments some of you more advanced programmers are making as you page through this book and look at the program listings. You are thinking that they will be somewhat inefficient to store and execute, the way I have all the statements laid out on separate lines. Yes, you are correct. These programs are not coded for efficiency. But there is a very good reason for this.

The purpose of this book is to teach you how Sprite graphics work. My main objective in including program listings is to give you easy-to-understand examples of what Sprites can do. And even though "crunched" programs (ones with all the spaces removed and absolutely everything that is unnecessary taken out) may be more efficient to run, they aren't very easy to read, or to understand.

There will be time later to crunch your own Sprite programs. (Consult the crunching guide on page 24 in the *Commodore 64 Programmer's Reference Guide* for some good suggestions.) In the meantime, I will write all my sample programs in a form that will be easiest for you to read.

If you've looked through the index, you see this book does not contain a section on how to do bit-mapped graphics, or any other fancy type of graphics other than Sprites. It also does not tell you how to add sound to your Sprite programs. The star of this show is Sprite graphics, folks. If I tried to include everything else, you'd need a wheelbarrow to carry this book home.

One last thing to remember before we start. Even though we will be working with eight Sprites, they are numbered zero through seven, not one through eight. This takes a bit of getting used to.

A SIMPLE SAMPLE SPRITE PROGRAM

I know you're anxious to have something to show on the screen, so type this program in and run it. If you haven't made any typographical errors in the DATA statements, you should see a yellow butterfly floating diagonally down your screen.

```
10 REM SAMPLE SPRITE PROGRAM
15 REM UNCRUNCHED LISTING FOR READABILITY
20 V=53248
22 PRINT CHR$(147):POKE V+21,0
25 POKE 53280,5:POKE 53281,1
30 POKE 2040,192
40 FOR Y = 0 TO 62
50 READ A:POKE (192*64)+Y,A
70 NEXT Y
75 POKE V+39,7
80 POKE V+23,1:POKE V+29,1
88 Y=50:X=24
90 POKE V+0,X  :POKE V+1,Y
95 POKE V+21,1
100 Y=Y+1:X=X+1
110 IF Y = 255 THEN Y = 1:X=1
130 GOTO 90
10665 REM BUTTERFLY FROM CHAPTER 2
10670 DATA 2,0,64,49,0,140
10680 DATA 120,129,30,252,66,63
10690 DATA 254,36,127,255,24,255
```

```
10700 DATA 255,153,255,255,219,255
10710 DATA 255,255,255,255,255,255
10720 DATA 255,255,255,255,255,255
10730 DATA 255,255,255,255,255,255
10740 DATA 255,255,255,255,255,255
10750 DATA 255,219,255,127,153,254
10760 DATA 63,24,252,30,24,120
10770 DATA 12,24,48
```

Program Highlights A guided tour of the highlights of this sample program will help you understand our objectives in the next ten chapters.

Line 20 — sets the location for the beginning of the video chip.

Line 22 — clears the screen and turns off any Sprites which might have been on the screen.

Line 25 — sets the screen background and border color codes.

Line 30 — tells the program where in memory we will store the DATA values for Sprite #0.

Lines 40 thru 70 — reads the 63 DATA values for this Sprite into the correct memory locations.

Line 75 — assigns the color code for yellow to Sprite #0.

Line 80 — expands Sprite #0 to twice its normal length and twice its normal width.

Line 88 — initializes the values of the X and Y coordinates.

Line 90 — sets the X and Y coordinates to position Sprite #0 on the screen.

Line 95 — turns on Sprite #0.

Line 100 — increments the X and Y coordinate values, so the Sprite will move across the screen.

Line 110 — sets the X and Y coordinate values back to 1 when Y reaches its maximum value of 255.

Line 130 — goes back to 90 to continue the Sprite's movement across the screen.

Lines 10670 thru 10770 — contain the numeric values used to define the shape of the Sprite.

CHAPTER 2

DESIGNING AND DEFINING YOUR SPRITES

A Sprite is a high-resolution graphics shape composed of a grid of tiny dots of light called PIXELS. These pixels light up on your television screen or monitor to form patterns and shapes. Each Sprite grid is 24 dots wide by 21 dots long, for a total of 504 dots. Each row in the Sprite is divided into three groups of eight dots, for a reason you will see in just a moment.

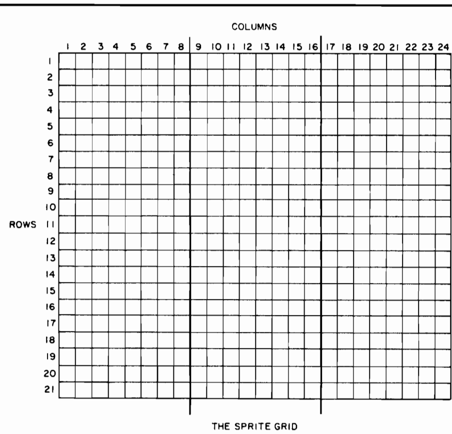
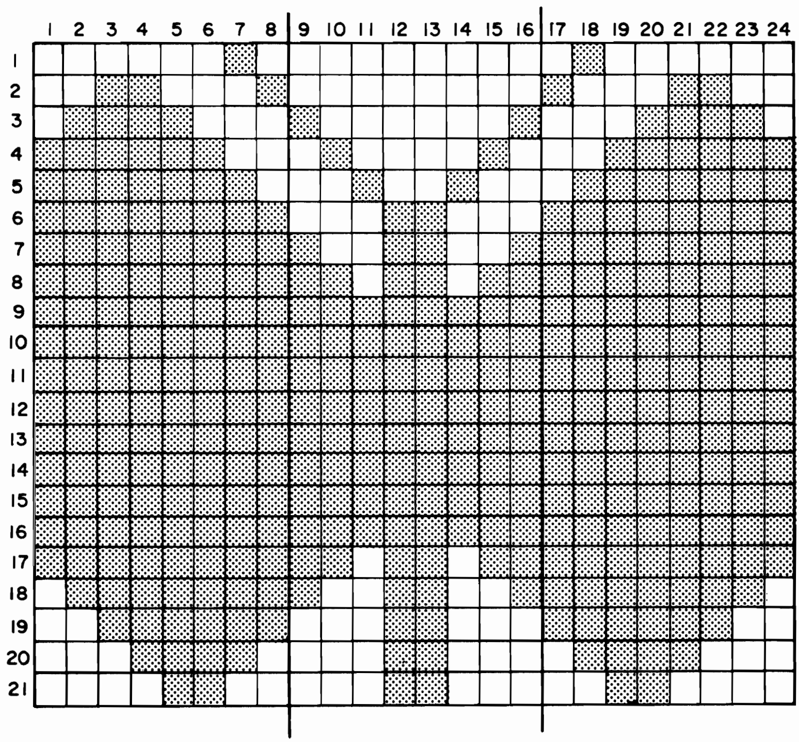


Figure 2-1

STARTING OFF

Your first task in designing a Sprite is to decide which of these dots you want "lighted" to form your Sprite shape. For these examples we will build single-color Sprites. Sprites can be made of up to four colors, using multi-color mode, but this is a bit more complicated and will be covered in Chapter 11.

In our example, the Sprite shape will be a butterfly. The shaded squares show the dots which will be lighted when the Sprite is shown on the screen.



BUTTERFLY SPRITE

Figure 2-2

Beginning Binaries

Once you have shaded in the Sprite shape you like, it has to be coded into a form the computer can understand. To do this, we convert each shaded dot on the grid into a "1" and make each empty dot a "0." Each row in the grid is then divided into three 8-digit binary numbers. In the butterfly example, the first row across now becomes:

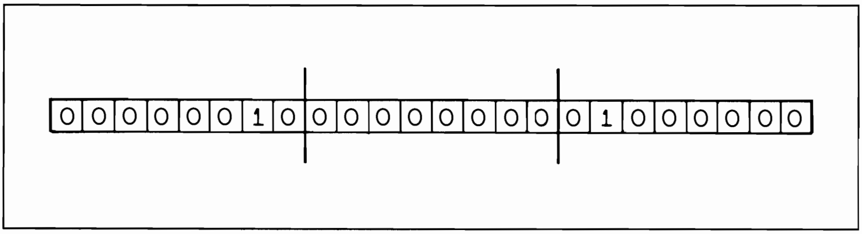


Figure 2-3

(If you need help with binary numbers, this is the time to read through Chapter 13.)

Each of these 8-digit binary numbers will be converted into their decimal equivalent. The resulting three decimal numbers will be used in DATA statements later in the program.

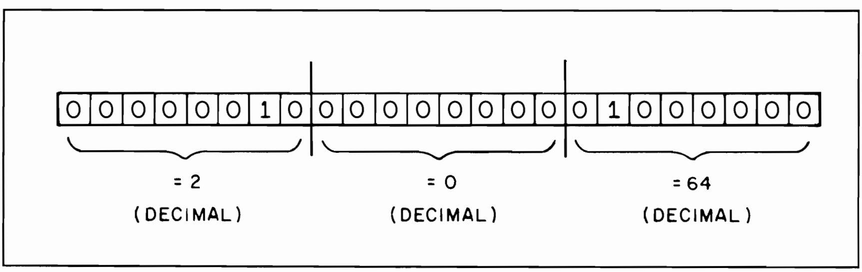


Figure 2-4

So the first line in the Sprite is converted into the following DATA statement:

```
10 DATA 2,0,64
```

We do the same for all 21 rows in the Sprite figure, and the DATA statements end up like this:

```
1000 REM BUTTERFLY SPRITE EXAMPLE
1010 DATA 2,0,64
1020 DATA 49,0,140
1030 DATA 120,129,30
1040 DATA 252,66,63
1050 DATA 254,36,127
1060 DATA 255,24,255
1070 DATA 255,153,255
1080 DATA 255,219,255
1090 DATA 255,255,255
1100 DATA 255,255,255
1110 DATA 255,255,255
1120 DATA 255,255,255
1130 DATA 255,255,255
1140 DATA 255,255,255
1150 DATA 255,255,255
1160 DATA 255,255,255
1170 DATA 255,219,255
1180 DATA 127,153,254
1190 DATA 63,24,252
1200 DATA 30,24,120
1210 DATA 12,24,48
```

You may be wondering why each data statement contains only three numbers. This was done strictly for readability, and so you can easily see which three numbers correspond to each line in the Sprite grid. When you write your own Sprite programs, you should put more numbers in each DATA statement, for program efficiency.

Making Your Binary Conversion Job Easier

Converting your Sprite drawing into these 63 data statements is the most time-consuming (and boring) part of working with Sprite graphics. One small mistake here, and your finished

Sprite can look totally unrecognizable. To make this process a bit easier, try these ideas:

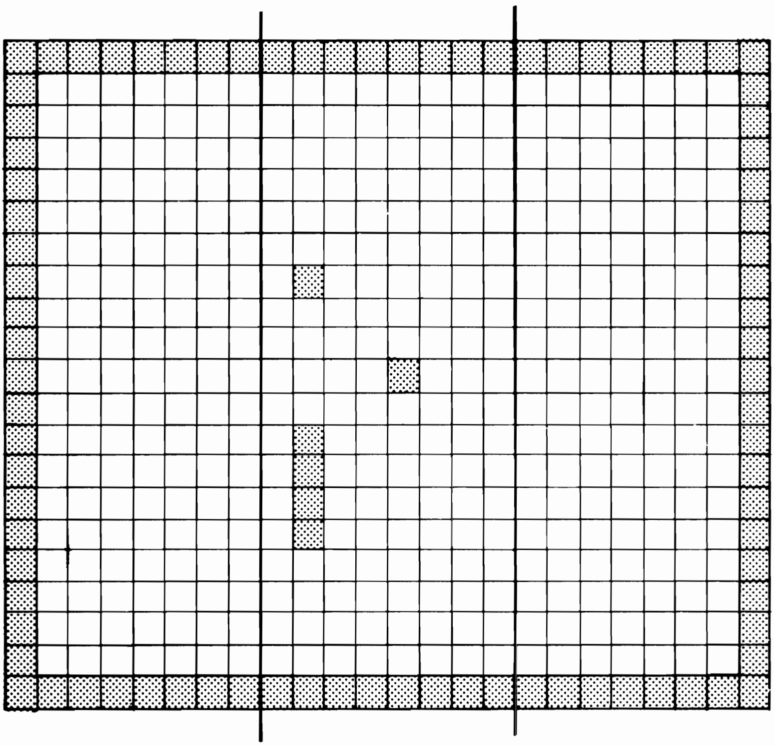
1. Use the binary conversion program given at the end of Chapter 6 (page 78) in your *Commodore 64 User's Guide*. Unless you speak fluent binary, mistakes can easily be made when you convert your binary numbers to their decimal values. This little program will be a big help.
2. Look for sections of your Sprite where the pattern repeats. Of course, lines which have areas of all blank dots, or areas with all 8 dots shaded in are easily recognized. But don't forget to look for other repeated patterns, too. If your shape is symmetrical, there may be many repeated areas. (In our butterfly example, Lines 8 and 17 are identical.)
3. Always count the number of decimal values you end up with. If you don't have 63 numbers, something is missing.
4. Put REMARK statements in with your DATA statements, showing where each new Sprite's DATA statements begin. It's much easier to trace mistakes later, especially if you know where each Sprite's 63 values begin.
5. Commercial software is available which allows you to "draw" Sprites on the screen, make any kind of changes, and then have the program translate the finished grid into DATA statements. Most of these packages allow you to copy and modify Sprites you have already done, change the colors to experiment with different effects, and view a series of Sprites in animation sequence. Some packages will even rotate a Sprite for you within the grid (make your original shape upside-down, for instance). If you plan to do a lot of Sprite work, the $30-40 cost is well worth it. I find the package I use to be especially helpful when I'm doing multi-color Sprites, or an animation sequence. You can also find programs in computer user magazines which do some of the same things. (See list at end of chapter.)

THINGS TO CONSIDER WHEN DESIGNING SPRITES

A 21 by 24 grid is not very large, if you are trying to design a complex Sprite. In the beginning, you will get better results if

you choose relatively simple shapes and make them fill as much of the grid as possible. Sprites designed in this fashion tend to show up better on the screen, and to work reasonably well with a wider range of screen/Sprite color combinations.

As you work more with Sprites, you will see that some colors "fight" each other on the screen, due to the way your television set or monitor produces different colors. When battling colors are placed next to each other, the edges may look blurred. This effect will be intensified if the Sprite involved is very small, or has small areas of color surrounded by large uncolored areas where the background shows through.



THE ISOLATED DOTS IN THE CENTER OF THIS SPRITE MAY NOT BE EASILY SEEN WITH SOME COLOR COMBINATIONS OF SPRITE AND SCREEN BACKGROUND.

Figure 2-5

Interesting results can be obtained by making the "picture" part of your Sprite out of the blank areas instead of the shaded areas.

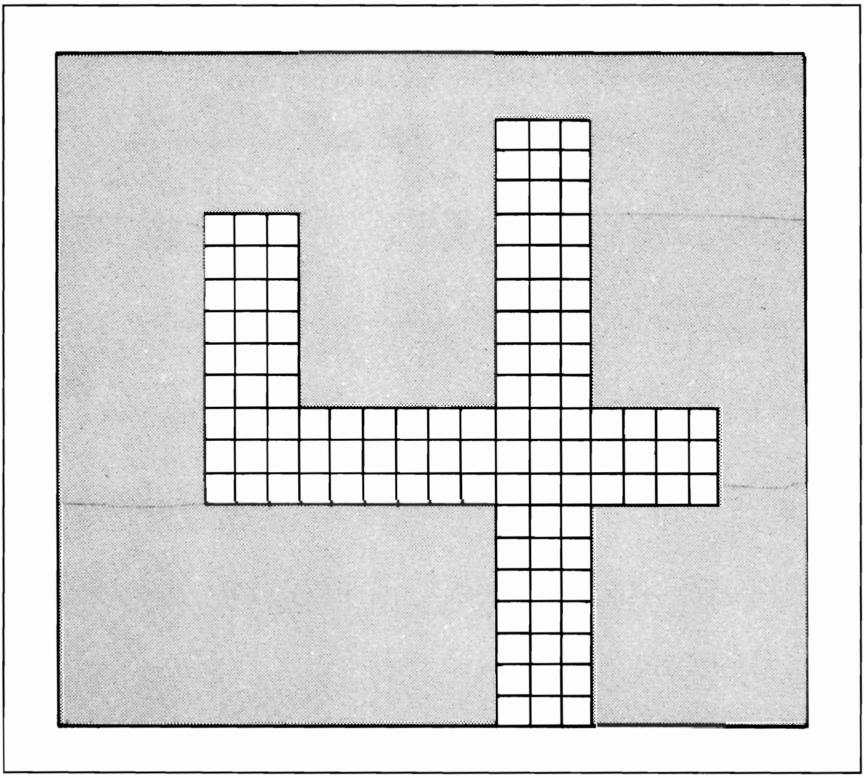


Figure 2-6

Making "Siamese" Sprites

A single Sprite can be defined no larger than 24 dots across by 21 dots down. However, if you need a larger grid to work with, there is a way around this limitation. You can design TWO Sprites that are constructed so that placed side by side on the screen, they will make one large figure. Here's an example.

You want to make a Sprite in the shape of a house, but the house you have in mind will not fit inside a single Sprite grid.

So you design two Sprites — one to be the left side of the house, the other to be the right side.

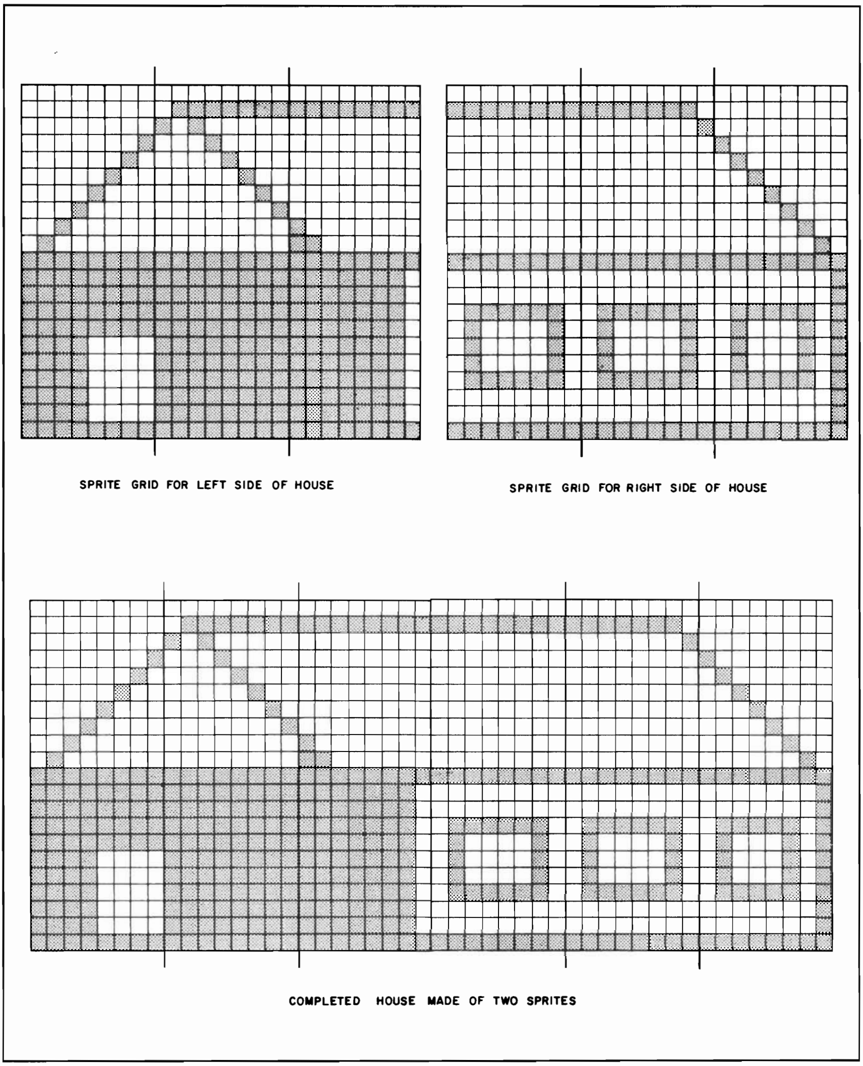


Figure 2-7

When you position these two Sprites on the screen, you place them so they touch and look like one large figure. As long as

you are certain to move them around together, no one will be the wiser!

As a matter of fact, your two Sprites wouldn't have to just touch each other — they could actually "interlock." In this example, the two Sprites would be assigned different colors. If they were positioned correctly, they would look like puzzle pieces being assembled.

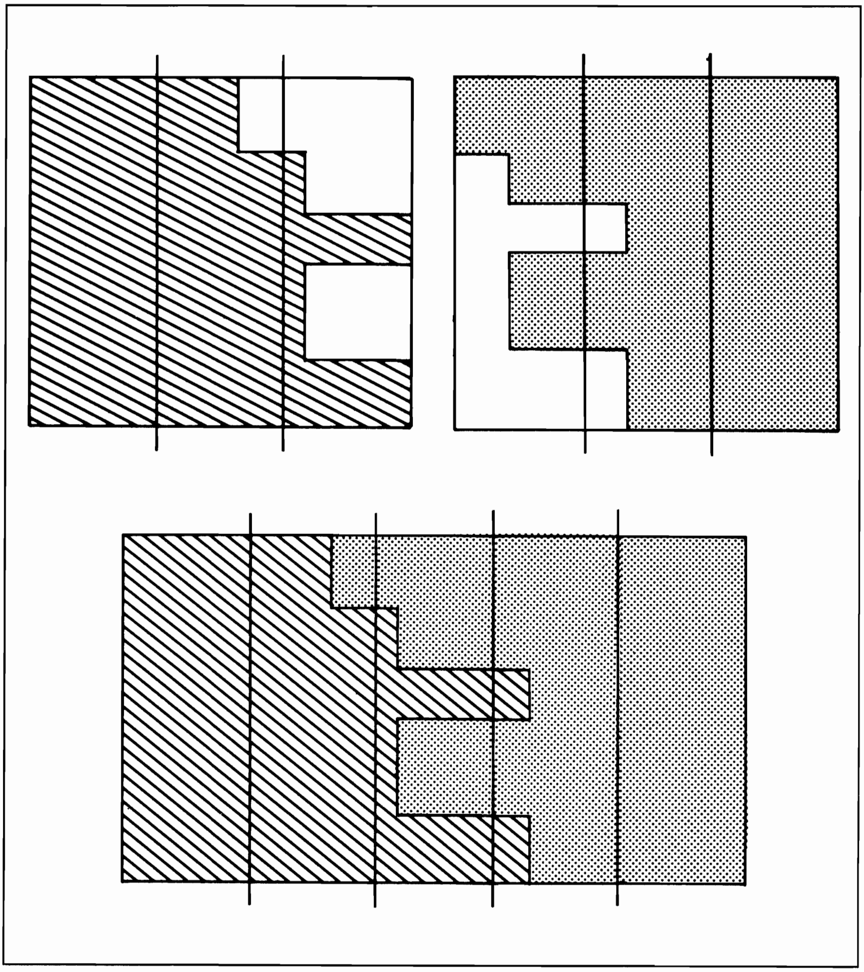


Figure 2-8

FINDING IDEAS FOR SPRITE SHAPES

Assuming you are not artistically inclined, where can you find helpful ideas for how to draw Sprite shapes?

One of the most obvious sources is to look over peoples' shoulders at the arcade. (Or you could make a great sacrifice and play a few games yourself.) You will be pleasantly surprised to realize that many of your favorite (and very visually exciting) games depend on simple shapes. There may be a lot going on in the background, with colors changing and lightning flashing as the universe tumbles around you, but the shape that the player is manipulating is probably quite simple. Spaceship Sprites are fun to do, because you really can't go wrong. Just about any space ship design will look great against an interesting graphics background.

It is relatively easy to find examples of typestyles for numerals and letters of the alphabet which can be made with mostly (or solely) straight lines.

Logos for products and companies can provide interesting shapes and designs. Look around you to see how advertisers can get an idea across by the suggestion of a shape.

A rather unlikely source, but one that can be helpful, is counted cross-stitch or needlepoint pattern books. These patterns are built on squares, and the smaller ones will translate quite nicely into Sprites. The only drawback is that many of them will require more squares than you will have room for on your 21 by 24 Sprite grid. You can find these books in hobby and needlework shops or at your public library. However, be prepared for strange looks from salespeople if you divulge your motivation for looking through books of simple patterns!

SAMPLE SPRITE VALUES

If you prefer to wait a bit before designing your own Sprite shapes, here are the values for some of the Sprites shown in the color illustrations for this book. All these examples are single-color Sprites. The multi-color Sprites are found in Chapter 11.

Outline Butterfly

This delicate Sprite has only an outline for a body. It does not work well when you are using Sprite and screen background colors which "fight" each other.

```
10135 REM OUTLINE BUTTERFLY
10140 DATA 3,0,192
10150 DATA 57,129,156,108,195
10160 DATA 54,198,102,99,131
10170 DATA 36,193,129,153,129
10180 DATA 128,219,1,128,126
10190 DATA 1,128,60,1,128
10200 DATA 24,1,128,24,1
10210 DATA 128,24,1,128,24
10220 DATA 1,128,24,1,128
10230 DATA 24,1,128,24,1
10240 DATA 128,126,3,96,219
10250 DATA 6,49,153,140,27
10260 DATA 24,216,14,24,112
```

Solid Butterfly with Straight Wings

This sturdy Sprite fills the Sprite grid completely from side to side. It looks especially nice when combined with the other types of butterfly Sprites.

```
10270 REM SOLID BUTTERFLY
10280 DATA 3,0,192,57,129
10290 DATA 156,124,195,62,254
10300 DATA 102,127,255,36,255
10310 DATA 255,153,255,255,219
10320 DATA 255,255,255,255,255
10330 DATA 255,255,255,255,255
10340 DATA 255,255,255,255,255
10350 DATA 255,255,255,255,255
10360 DATA 255,255,255,255,255
10370 DATA 255,255,255,255,255
10380 DATA 255,127,219,254,63
10390 DATA 153,252,31,24,248
10400 DATA 14,24,112
```

Solid Butterfly with Notched Wings

Of all the butterfly Sprites, this is my masterpiece. It looks the most like a real butterfly, and adds a bright touch to the screen, since it can be defined in many different colors.

```
10405 REM NOTCHED WING BUTTERFLY
10410 DATA 3,0
10420 DATA 192,57,129,156,124
10430 DATA 195,62,254,102,127
10440 DATA 255,36,255,255,153
10450 DATA 255,127,219,254,63
10460 DATA 219,252,31,219,248
10470 DATA 15,219,240,15,219
10480 DATA 240,31,219,248,63
10490 DATA 219,252,127,219,254
10500 DATA 255,219,255,255,219
10510 DATA 255,255,219,255,127
10520 DATA 219,254,63,153,252
10530 DATA 31,24,248,14,24
10540 DATA 112
```

Solid Fish

I wouldn't look for a fish like this in your aquarium, but it is recognizable. This Sprite can be seen swimming underwater in the color illustration of the lake scene.

```
10545 REM SOLID FISH
10550 DATA 0,0,0,0
10560 DATA 0,0,0,0,0
10570 DATA 0,0,0,3,240
10580 DATA 0,7,248,0,15
10590 DATA 252,3,28,254,3
10600 DATA 60,255,15,255,255
10610 DATA 254,31,255,252,31
10620 DATA 255,254,255,255,15
10630 DATA 31,254,3,15,252
10640 DATA 1,7,248,0,3
10650 DATA 240,0,0,0,0
10660 DATA 0,0,0,0,0
10670 DATA 0,0,0,0
```

Jellyfish/Octopus

I couldn't make this Sprite look much like either a jellyfish or an actual octopus, but he did turn out to be rather cute. You can also see him in the colored illustration of the lake scene.

```
10675 REM JELLYFISH
10680 DATA 3
10690 DATA 255,128,7,255,192
10700 DATA 15,24,224,31,255
10710 DATA 240,60,0,120,60
10720 DATA 0,120,63,255,248
10730 DATA 6,170,192,12,170
10740 DATA 96,25,170,48,51
10750 DATA 43,24,102,109,140
10760 DATA 204,196,196,204,198
10770 DATA 198,204,198,198,204
10780 DATA 198,198,204,198,198
10790 DATA 204,198,198,204,198
10800 DATA 198,204,198,198,204
10810 DATA 198,198
```

Man Wearing Swim Suit

Yes, that is a tiny little bellybutton you see, if you look closely at this Sprite. He can be seen standing on the dock in the colored illustration of the lake scene.

```
10950 REM MAN IN SWIMSUIT
10960 DATA 0,62,0,0,42
10970 DATA 0,0,62,0,0
10980 DATA 28,0,0,127,0
10990 DATA 1,255,192,3,62
11000 DATA 96,6,62,48,28
11010 DATA 54,56,8,62,16
11020 DATA 0,0,0,0,62
11030 DATA 0,0,62,0,0
11040 DATA 54,0,0,0,0
11050 DATA 0,54,0,0,54
11060 DATA 0,0,54,0,0
11070 DATA 54,0,0,54,0
11080 DATA 1,247,128
```

Woman Wearing Swim Suit

Her hairdo is a bit outdated, but you can definitely tell she's female. She and her male friend were designed to be about the same size. When you move them towards each other horizontally, their hands are the first points to touch.

```
11085 REM WOMAN IN SWIM SUIT
11090 DATA 0,62
11100 DATA 0,0,170,128,0
11110 DATA 255,128,0,28,0
11120 DATA 0,127,0,1,193
11130 DATA 192,3,54,96,6
11140 DATA 62,48,28,62,56
11150 DATA 8,62,16,0,127
11160 DATA 0,0,255,128,0
11170 DATA 0,0,0,54,0
11180 DATA 0,54,0,0,54
11190 DATA 0,0,54,0,0
11200 DATA 54,0,0,54,0
11210 DATA 0,54,0,1,247
11220 DATA 128
```

Inverse Letter "G"

The actual letter in the Sprite is formed by the transparent area which lets the background show through. The surrounding color fills the Sprite to all four edges.

```
9999 REM LETTER G
10000 DATA 255
10010 DATA 255,255,255,255,255
10020 DATA 224,0,7,224,0
10030 DATA 7,231,255,231,231
10040 DATA 255,231,231,255,255
10050 DATA 231,255,255,231,255
10060 DATA 255,231,255,255,231
10070 DATA 255,255,231,240,7
10080 DATA 231,240,7,231,255
10090 DATA 231,231,255,231,231
10100 DATA 255,231,231,255,231
10110 DATA 224,0,7,224,0
```

```
10120 DATA 7,255,255,255,255
10130 DATA 255,255
```

Inverse Letter "O"

The actual letter in this Sprite is formed by the transparent area which lets the background show through. It could also be used for the number zero.

```
10135 REM LETTER O
10140 DATA 255,255,255
10150 DATA 255,255,255,224,0
10160 DATA 7,224,0,7,224
10170 DATA 0,7,227,255,199
10180 DATA 227,255,199,227,255
10190 DATA 199,227,255,199,227
10200 DATA 255,199,227,255,199
10210 DATA 227,255,199,227,255
10220 DATA 199,227,255,199,227
10230 DATA 255,199,227,255,199
10240 DATA 224,0,7,224,0
10250 DATA 7,224,0,7,255
10260 DATA 255,255,255,255,255
```

Inverse Letter "D"

The actual letter in this Sprite is formed by the transparent area which lets the background show through. It is very striking when used against a dark background instead of a light one.

```
10270 REM LETTER D
10280 DATA 255,255,255,255,255
10290 DATA 255,224,0,7,224
10300 DATA 0,7,249,255,231
10310 DATA 249,255,231,249,255
10320 DATA 231,249,255,231,249
10330 DATA 255,231,249,255,231
10340 DATA 249,255,231,249,255
10350 DATA 231,249,255,231,249
10360 DATA 255,231,249,255,231
```

```
10370 DATA 249,255,231,249,255
10380 DATA 231,224,0,7,224
10390 DATA 0,7,255,255,255
10400 DATA 255,255,255
```

Inverse Letter "J"

The actual letter in this Sprite is formed by the transparent area which lets the background show through. Check the inverse Sprite programs in Chapter 12 if you would like to manipulate this Sprite's data to produce a normal letter.

```
10405 REM LETTER J
10410 DATA 255,255
10420 DATA 255,255,255,255,224
10430 DATA 0,7,224,0,7
10440 DATA 255,227,255,255,227
10450 DATA 255,255,227,255,255
10460 DATA 227,255,255,227,255
10470 DATA 255,227,255,255,227
10480 DATA 255,255,227,255,231
10490 DATA 227,255,231,227,255
10500 DATA 231,227,255,231,227
10510 DATA 255,231,227,255,224
10520 DATA 3,255,224,3,255
10530 DATA 255,255,255,255,255
10540 DATA 255
```

Inverse Letter "B"

The actual letter in this Sprite is formed by the transparent area which lets the background show through. Having the entire alphabet of Sprite letters saved for later use will come in very handy if you are designing letter-recognition games for your preschoolers.

```
10545 REM LETTER B
10550 DATA 255,255,255,255
10560 DATA 255,255,192,0,3
10570 DATA 192,0,3,243,255
10580 DATA 243,243,255,243,243
```

```
10590 DATA 255,243,243,255,243
10600 DATA 243,255,243,240,0
10610 DATA 3,240,0,31,240
10620 DATA 0,3,243,255,243
10630 DATA 243,255,243,243,255
10640 DATA 243,243,255,243,243
10650 DATA 255,243,192,0,3
10660 DATA 192,0,3,255,255
10670 DATA 255,255,255,255
```

Inverse Exclamation Point

This inverse Sprite is particularly colorful, since the transparent area is so small compared to the colored background which fills the rest of the Sprite grid.

```
10675 REM EXCLAMATION POINT
10680 DATA 255
10690 DATA 255,255,255,255,255
10700 DATA 255,231,255,255,231
10710 DATA 255,255,231,255,255
10720 DATA 231,255,255,231,255
10730 DATA 255,231,255,255,231
10740 DATA 255,255,231,255,255
10750 DATA 231,255,255,231,255
10760 DATA 255,231,255,255,231
10770 DATA 255,255,231,255,255
10780 DATA 231,255,255,255,255
10790 DATA 255,231,255,255,231
10800 DATA 255,255,255,255,255
10810 DATA 255,255
```

Letter "E"

This is a normal Sprite, with the colored portion of the Sprite making up the actual letter.

```
10815 REM LETTER E
10820 DATA 255,255,255
10830 DATA 255,255,255,192,0
10840 DATA 3,192,0,3,192
```

```
10850 DATA 0,3,192,0,0
10860 DATA 192,48,0,192,48
10870 DATA 0,192,48,0,255
10880 DATA 240,0,255,240,0
10890 DATA 192,48,0,192,48
10900 DATA 0,192,48,0,192
10910 DATA 0,0,192,0,3
10920 DATA 192,0,3,192,0
10930 DATA 3,192,0,3,255
10940 DATA 255,255,255,255,255
```

Letter "F"

If you have a Sprite software package or program to help you design Sprites quickly and easily, try doing the entire alphabet and storing it on a disk or tape. This set of Sprites can be used over and over for a variety of educational programs.

```
10950 REM LETTER F
10960 DATA 255,255,224,255,255
10970 DATA 224,192,0,0,192
10980 DATA 0,0,192,0,0
10990 DATA 192,0,0,192,0
11000 DATA 0,192,48,0,192
11010 DATA 48,0,255,240,0
11020 DATA 255,240,0,192,48
11030 DATA 0,192,48,0,192
11040 DATA 0,0,192,0,0
11050 DATA 192,0,0,192,0
11060 DATA 0,192,0,0,192
11070 DATA 0,0,192,0,0
11080 DATA 192,0,0
```

Letter "G"

You can make an "inverse" Sprite out of this normal letter by using a simple technique found in Chapter 12.

```
11085 REM LETTER G
11090 DATA 63,255
11100 DATA 252,63,255,252,63
```

```
11110 DATA 255,252,56,0,28
11120 DATA 56,0,28,56,0
11130 DATA 0,56,0,0,56
11140 DATA 0,0,56,0,0
11150 DATA 56,0,0,56,15
11160 DATA 252,56,15,252,56
11170 DATA 15,252,56,0,28
11180 DATA 56,0,28,56,0
11190 DATA 28,56,0,28,56
11200 DATA 0,28,63,255,252
11210 DATA 63,255,252,63,255
11220 DATA 252
```

Letter "H"

You will quickly discover that the easiest letters to design are those which have all straight lines.

```
11225 REM LETTER H
11230 DATA 56,0,112,56
11240 DATA 0,112,56,0,112
11250 DATA 56,0,112,56,0
11260 DATA 112,56,0,112,56
11270 DATA 0,112,56,0,112
11280 DATA 56,0,112,63,255
11290 DATA 240,63,255,240,56
11300 DATA 0,112,56,0,112
11310 DATA 56,0,112,56,0
11320 DATA 112,56,0,112,56
11330 DATA 0,112,56,0,112
11340 DATA 56,0,112,56,0
11350 DATA 112,56,0,112
```

Letter "I"

This letter is so easy to design that you hardly need to draw it on graph paper, if you stop and think about it for a minute.

```
11355 REM LETTER I
11360 DATA 255
11370 DATA 255,255,255,255,255
```

```
11380 DATA 255,255,255,0,24
11390 DATA 0,0,24,0,0
11400 DATA 24,0,0,24,0
11410 DATA 0,24,0,0,24
11420 DATA 0,0,24,0,0
11430 DATA 24,0,0,24,0
11440 DATA 0,24,0,0,24
11450 DATA 0,0,24,0,0
11460 DATA 24,0,0,24,0
11470 DATA 0,24,0,255,255
11480 DATA 255,255,255,255,255
11490 DATA 255,255
```

Letter "J"

Save yourself some work by adapting existing Sprites to make new ones. For instance, my letter "J" is nothing more than the letter "I" with a tail added to it. This saves me having to code about three-fourths of the data values again.

```
11495 REM LETTER J
11500 DATA 255,255,255
11510 DATA 255,255,255,255,255
11520 DATA 255,0,30,0,0
11530 DATA 30,0,0,30,0
11540 DATA 0,30,0,0,30
11550 DATA 0,0,30,0,0
11560 DATA 30,0,0,30,0
11570 DATA 0,30,0,224,30
11580 DATA 0,224,30,0,224
11590 DATA 30,0,224,30,0
11600 DATA 224,30,0,224,30
11610 DATA 0,255,254,0,255
11620 DATA 254,0,255,254,0
```

Letter "K"

Of all the alphabet letters, this has to be one of the worst to code as a Sprite. Be sure to make it occupy as much of the Sprite grid as you can, so you have more room to spread the diagonal lines.

```
11630 REM LETTER K
11640 DATA 224,7,0,224,14
11650 DATA 0,224,28,0,224
11660 DATA 56,0,224,112,0
11670 DATA 224,224,0,225,192
11680 DATA 0,227,128,0,231
11690 DATA 0,0,254,0,0
11700 DATA 252,0,0,254,0
11710 DATA 0,231,0,0,227
11720 DATA 128,0,225,192,0
11730 DATA 224,224,0,224,112
11740 DATA 0,224,56,0,224
11750 DATA 28,0,224,14,0
11760 DATA 224,7,0
```

Inverse Question Mark

This transparent question mark against a colored background is handy for lots of programming applications. It shows up well in normal size or when expanded in either or both directions.

```
11765 REM QUESTION MARK
11770 DATA 255,255
11780 DATA 255,252,0,15,248
11790 DATA 255,143,241,255,143
11800 DATA 227,255,143,231,255
11810 DATA 15,227,254,31,225
11820 DATA 252,63,240,248,127
11830 DATA 252,240,255,255,225
11840 DATA 255,255,195,255,255
11850 DATA 195,255,255,195,255
11860 DATA 255,195,255,255,195
11870 DATA 255,255,195,255,255
11880 DATA 255,255,255,195,255
11890 DATA 255,195,255,255,255
11900 DATA 255
```

Starfighter with Straight Wings

This Sprite is simple, but very effective. I've used it in several action games.

```
9999 REM STARFIGHTER
10000 DATA 0
10010 DATA 0,0,0,0,0
10020 DATA 0,0,0,0,0
10030 DATA 0,0,0,0,0
10040 DATA 0,0,129,255,129
10050 DATA 131,255,193,135,255
10060 DATA 225,135,255,225,255
10070 DATA 255,255,131,255,193
10080 DATA 129,255,129,128,0
10090 DATA 1,128,0,1,0
10100 DATA 0,0,0,0,0
10110 DATA 0,0,0,0,0
10120 DATA 0,0,0,0,0
10130 DATA 0,0
```

Starfighter with Folded Wings

A variation on the first starfighter, this Sprite shows new action in my "Conquer the Universe" games. It is essentially the same size as the original starfighter.

```
10135 REM STARFIGHTER WITH FOLDED WINGS
10140 DATA 0,0,0
10150 DATA 0,0,0,0,0
10160 DATA 0,0,0,0,0
10170 DATA 0,0,0,0,0
10180 DATA 17,255,136,35,255
10190 DATA 196,71,255,226,135
10200 DATA 255,225,255,255,255
10210 DATA 131,255,193,65,255
10220 DATA 130,32,0,4,16
10230 DATA 0,8,0,0,0
10240 DATA 0,0,0,0,0
10250 DATA 0,0,0,0,0
10260 DATA 0,0,0,0,0
```

Rocket Pointing Upwards

This Sprite has a striped body (alternating columns of color and background). It looks very different depending on what type of monitor or television set you are using. It also changes

in appearance depending on what Sprite and screen color combination you choose.

```
10405 REM SHAPELY ROCKET
10410 DATA 0,16
10420 DATA 0,0,84,0,1
10430 DATA 85,0,5,85,64
10440 DATA 21,85,80,21,85
10450 DATA 80,21,85,80,21
10460 DATA 85,80,21,85,80
10470 DATA 21,85,80,21,85
10480 DATA 80,21,85,80,21
10490 DATA 85,80,16,16,16
10500 DATA 16,16,16,16,0
10510 DATA 16,16,0,16,16
10520 DATA 0,16,16,0,16
10530 DATA 16,0,16,16,0
10540 DATA 16
```

Rocket Blasting Off in Upward Direction

This is the same rocket, but with exhaust fumes coming out of the tail. However, if you choose color combinations which "fight" on the screen, the exhaust pixels won't be visible at all!

```
10545 REM SHAPELY ROCKET PLUS EXHAUST
10550 DATA 0,16,0,0
10560 DATA 84,0,1,85,0
10570 DATA 5,85,64,21,85
10580 DATA 80,21,85,80,21
10590 DATA 85,80,21,85,80
10600 DATA 21,85,80,21,85
10610 DATA 80,21,85,80,21
10620 DATA 85,80,21,85,80
10630 DATA 16,16,16,16,16
10640 DATA 16,16,0,16,18
10650 DATA 68,144,16,0,16
10660 DATA 18,68,144,16,0
10670 DATA 16,18,68,144
```

Coffee Cup

This deceptively simple Sprite is one of my favorites. Depending on where you place it on the screen and what colors you choose, the steam rising from the cup will appear at times to be multi-colored, even though this is a single color Sprite.

```
10675 REM COFFEE CUP
10680 DATA 8
10690 DATA 66,0,4,33,0
10700 DATA 2,16,128,4,33
10710 DATA 0,8,66,0,0
10720 DATA 0,0,255,255,192
10730 DATA 255,255,192,245,85
10740 DATA 252,245,85,254,245
10750 DATA 85,199,245,85,195
10760 DATA 245,85,195,245,85
10770 DATA 195,245,85,199,245
10780 DATA 85,206,245,85,252
10790 DATA 245,85,248,245,85
10800 DATA 240,245,85,192,255
10810 DATA 255,192
```

Empty Zoo Cage/Jail

This Sprite is best defined as #0, since you will usually want to put it on top of other Sprites, so it will look like the background Sprite is in jail or in a cage at the zoo. In Chapter 4 you will learn that Sprites with the lowest numbers have the highest priority, when Sprites meet each other on the screen.

```
10815 REM ZOO CAGE
10820 DATA 255,255,255
10830 DATA 255,255,255,132,16
10840 DATA 65,132,16,65,132
10850 DATA 16,65,132,16,65
10860 DATA 132,16,65,132,16
10870 DATA 65,132,16,65,132
10880 DATA 16,65,132,16,65
10890 DATA 132,16,65,132,16
10900 DATA 65,132,16,65,132
10910 DATA 16,65,132,16,65
```

```
10920 DATA 132,16,65,132,16
10930 DATA 65,132,16,65,255
10940 DATA 255,255,255,255,255
```

Happy Rabbit

This cartoon-type rabbit is useful for chase scenes and perking up instructional programs. However, if you define him in a color that is not compatible with the background colors, you may see "sparkles" of light in his eyes as you move him around the screen.

```
9999 REM SPRITE DATA FOR HAPPY RABBIT
10000 DATA 0
10010 DATA 0,0,255,0,255
10020 DATA 255,129,255,15,199
10030 DATA 224,3,199,128,0
10040 DATA 238,0,0,238,0
10050 DATA 0,238,0,0,255
10060 DATA 0,1,255,128,3
10070 DATA 147,192,7,147,224
10080 DATA 15,147,240,31,239
10090 DATA 248,31,125,248,15
10100 DATA 131,240,7,131,224
10110 DATA 3,131,192,1,255
10120 DATA 128,0,127,0,0
10130 DATA 0,0
```

Surprised Rabbit

This alternate version of the rabbit is used in the Spritesampler program found in Chapter 12. These two views are rapidly alternated as the wolf chases the rabbit across the screen.

```
10135 REM SPRITE DATA FOR SURPRISED RABBIT
10140 DATA 0,0,0
10150 DATA 0,0,0,0,0
10160 DATA 0,31,199,240,27
10170 DATA 199,176,24,238,48
10180 DATA 24,238,48,24,238
10190 DATA 48,24,255,48,25
```

```
10200 DATA 255,176,3,147,192
10210 DATA 7,147,224,15,147
10220 DATA 240,31,239,248,31
10230 DATA 255,248,15,255,240
10240 DATA 7,199,224,3,199
10250 DATA 192,1,199,128,0
10260 DATA 127,0,0,0,0
```

Wolf

Defining this Sprite in a color that "fights" with the background is particularly amusing. As the wolf moves across the screen, his teeth look as if they are sparkling! (This may not happen with your television or monitor, but it does every time with mine.)

```
10270 REM SPRITE DATA FOR WOLF
10280 DATA 0,0,0,0,0
10290 DATA 0,0,0,48,1
10300 DATA 192,112,0,240,240
10310 DATA 0,121,224,0,61
10320 DATA 192,0,31,192,0
10330 DATA 15,224,0,63,224
10340 DATA 0,255,240,127,199
10350 DATA 248,255,255,252,255
10360 DATA 255,254,255,255,254
10370 DATA 128,255,254,42,255
10380 DATA 254,127,255,252,1
10390 DATA 255,248,1,255,240
10400 DATA 1,255,240
```

Mean Wolf

This is the alternative version of the wolf who chases the rabbit in the Spritesampler program in Chapter 12. If you need a wolf who is facing right instead of left (as this one is), try the Sprite reversal routines found in Chapter 14. You'll get two Sprites for the price of one, without having to design and code the second Sprite.

```
10405 REM SPRITE DATA FOR MEAN WOLF
10410 DATA 0,0
10420 DATA 0,0,0,0,0
10430 DATA 0,48,1,192,112
10440 DATA 0,240,240,0,121
10450 DATA 224,0,61,192,0
10460 DATA 31,192,0,15,224
10470 DATA 0,63,224,255,255
10480 DATA 240,255,193,248,255
10490 DATA 255,252,170,255,254
10500 DATA 0,255,254,0,255
10510 DATA 254,0,255,254,170
10520 DATA 255,252,255,255,248
10530 DATA 1,255,240,1,255
10540 DATA 240
```

Anchor

The Navy would laugh if they saw this anchor, but it looks fine next to my crazy octopus and fish in the underwater lake scene in the color illustrations.

```
11225 REM ANCHOR
11230 DATA 0,30,0,0
11240 DATA 18,0,0,30,0
11250 DATA 0,12,0,0,12
11260 DATA 0,0,12,0,0
11270 DATA 12,0,12,12,24
11280 DATA 30,12,60,45,12
11290 DATA 90,12,12,24,12
11300 DATA 12,24,12,12,24
11310 DATA 12,12,24,12,12
11320 DATA 24,15,255,248,7
11330 DATA 255,240,0,12,0
11340 DATA 0,12,0,0,0
11350 DATA 0,0,0,0
```

Space Cruiser

You always need different shapes of spaceships to be the "good guys" and the "bad guys" in a game program, and this one can fit either bill.

```
10399 REM SPACE CRUISER
10400 DATA 64,146,44
10410 DATA 0,16
10420 DATA 0,0,84,0,1
10430 DATA 85,0,5,85,64
10440 DATA 21,85,80,21,85
10450 DATA 80,21,85,80,21
10460 DATA 85,80,21,85,80
10470 DATA 21,85,80,21,85
10480 DATA 80,21,85,80,21
10490 DATA 85,80,16,16,16
10500 DATA 16,16,16,16,0
10510 DATA 16,16,0,16,16
10520 DATA 0,16,16,0,16
10530 DATA 16,0,16,16,0
```

MAGAZINE ARTICLES OF INTEREST

"A Shape Generator for the Commodore 64", Donald A. Pitts, *COMPUTE!*, November, 1982, pp. 160-163.

"Commodore 64 Sprite Editor," Stephen Meirosky, *COMPUTE!*, December, 1982, pp. 212+.

"A Sprite Editor for the Commodore 64," John Michael Lane, *Creative Computing*, September, 1983, pp. 290-293.

"How to Create Your Own Sprite Creator," Tim Villaneuva, *COMMODORE POWER/PLAY*, Summer 1983, pp. 67-68.

CHAPTER 3

CHOOSING COLORS FOR YOUR SPRITES

Each Sprite has its own register in the video chip to denote the color that Sprite will be. The colors are coded by number. These are the 16 color codes available:

SPRITE COLOR CODES	
0 — black	8 — orange
1 — white	9 — brown
2 — red	10 — light red
3 — cyan (light blue)	11 — dark gray
4 — purple	12 — medium gray
5 — green	13 — light green
6 — blue	14 — light blue
7 — yellow	15 — light gray

PEEKING AT POKES

A color code is assigned to a Sprite by poking a value into a register. If you have been following this book sequentially, this is the first time we have used the POKE statement. For those of

you not familiar with it, let's digress a moment and discuss how it works.

In BASIC, there are two ways to store data in memory. One way is to set up a variable name for a memory location, and assign a data value to that location:

```
10 LET X=89
```

In this case, we only refer to the memory location containing the value 89 as "X." The computer keeps track of the actual memory location where this value is stored, by means of an internal table. But you as the programmer do not know where the data in X is actually stored in the memory.

ASSIGNING SPRITE COLORS

When we use the POKE statement, we give the actual memory address where the data should be stored. For instance, to assign the color orange to the screen border, we use the following POKE statement:

```
10 POKE 53280,8
```

This puts the code for the color orange into the register located at 53280.

To change the color of the screen background, we use:

```
10 POKE 53281,9
```

This pokes the color code for the color brown into the register at 53281.

Each Sprite (numbered from #0 thru 7) has its own register for a color code. Since these registers are all in the video chip,

we can simplify things by expressing the register number as an offset from the beginning address of the Video Chip, 53248. Thus, instead of having to memorize long numbers for each register's location, we just have to remember the offset number from the beginning address of the video chip.

OFFSET NUMBERS FOR REGISTER LOCATIONS

Sprite Number	*Color Register*
0	v + 39
1	v + 40
2	v + 41
3	v + 42
4	v + 43
5	v + 44
6	v + 45
7	v + 46

The color for each Sprite can be assigned individually:

```
5 V=53248
10 POKE V+24,4
20 POKE V+43,6
30 POKE V+39,2
40 POKE V+40,4
```

or if several Sprites will be the same color, we can take advantage of the fact that the color registers are in numeric sequence, and assign the color with a loop:

```
5 V=53248
10 FOR R=0 TO 7
20 POKE V+39+R,4
30 NEXT R
```

The example above will set all the color codes to purple. Notice that we must define the value for V early in each Sprite program, to specify the beginning address for the video chip.

If we want to assign a numeric sequence of color codes to the eight Sprites, we can also use a variable for the color code:

```
5 V=53248
10 FOR R=0 TO 7
20 POKE V+39+R,2+R
30 NEXT R
```

This will assign color codes 2 through 9 to Sprites 0 through 7.

Changing Color Assignments

It doesn't matter if you assign a color to a Sprite you end up not using in your program. Assigning a color to an unused Sprite will not cause any kind of error.

But you will have problems if you inadvertently assign a color code larger than 255 to a Sprite. A register cannot contain a number larger than 255, and you will get an "Illegal Quantity" error when you run the program. For example, this statement would cause an error:

```
10 POKE V+39,355
```

Colors for Sprites can be changed at any time during your program. Since POKE statements can also be used in direct mode, once you have Sprites displayed on the screen, you can POKE new values for Sprite colors and see immediate results.

Try It Yourself

In the sample Butterfly Program from Chapter 1, we defined the butterfly as Sprite #0, and gave it a color code of 7 (yellow). Load the Butterfly program into your computer and run it. While the butterfly is visible on the screen, stop the program. (Press RUN/STOP.) Then try these commands to change the color of the butterfly.

POKE V+39,2 will set the color to RED
POKE V+39,11 will set the color to DARK GRAY
POKE V+39,8 will set the color to ORANGE

CHAPTER 4

STORING SPRITES IN MEMORY

The Sprite shapes have been defined by converting them into DATA statements containing 63 numbers. Now these numbers must be read from the DATA statements and poked into designated locations in the computer's memory.

Storing Sprites in memory involves two separate tasks:

1. Telling the program *where* in the memory the data values for each Sprite will be stored.
2. Poking the values for each Sprite into the correct memory locations.

POINTING TO MEMORY LOCATIONS

Each Sprite has 63 data values associated with it, and one extra byte will be used as a placeholder to give a total of 64 bytes of memory needed for each Sprite's DATA values.

Each of the eight Sprites has a special memory location called a Sprite Pointer. This Sprite Pointer location is something like the Table of Contents for a book. The Table of Contents tells you where each chapter of a book begins. The Sprite pointer

tells the program where in memory the DATA values for each Sprite begin.

The pointers for the Sprites are at the locations shown in the table below. Notice that these memory locations are not inside the video chip, so we do not express them as an offset from the beginning of the video chip address.

SPRITE MEMORY LOCATIONS	
Sprite Number	*Pointer Location*
0	2040
1	2041
2	2042
3	2043
4	2044
5	2045
6	2046
7	2047

POKING PROPER VALUES

Now, what sort of number do we poke into the pointers to indicate where the Sprite data will be stored in the memory?

For this application, memory can be divided into areas of 64 bytes, just the size needed to hold the data for one Sprite. (Remember — each Sprite needs 63 bytes plus an extra unused byte as a placeholder.) Thus, areas of 64 bytes in memory are numbered, starting with area zero.

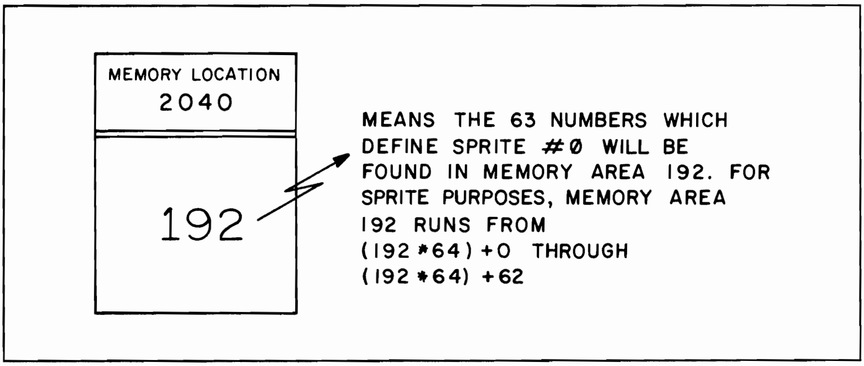


Figure 4-1

However, if you refer to the memory maps in your Commodore 64 reference manuals, you will see that you cannot poke DATA values into just any old place in the memory. Some of that memory is reserved for other special purposes, such as BASIC, or the operating system, or even the program you yourself are writing. Therefore, the chart below contains suggested memory areas for storing Sprite data. Using these areas, you will probably not get yourself into trouble for quite awhile, since by starting with memory area 192, you are leaving room for quite a sizeable BASIC program, yet you will have space for defining 64 different Sprites.

SPRITE NUMBER	POINTER LOCATION	SUGGESTED VALUE	POKE DATA VALUES FOR THE SPRITE INTO THESE 63 MEMORY LOCATIONS (REMEMBER-64th BIT OF EACH MEMORY AREA IS LEFT EMPTY AS A PLACE HOLDER.)
0	2040	192	(192*64)+0 THRU (192*64)+62
1	2041	193	(193*64)+0 THRU (193*64)+62
2	2042	194	(194*64)+0 THRU (194*64)+62
3	2043	195	(195*64)+0 THRU (195*64)+62
4	2044	196	(196*64)+0 THRU (196*64)+62
5	2045	197	(197*64)+0 THRU (197*64)+62
6	2046	198	(198*64)+0 THRU (198*64)+62
7	2047	199	(199*64)+0 THRU (199*64)+62

SPRITE POINTER SUMMARY

Figure 4-2

By setting the Sprite pointer for Sprite #0 to location 92, the data values for Sprite #0 must be read into memory at locations (192*64)+0 through (192*64)+62. This can most easily be accomplished with a FOR-NEXT loop:

```
10 V=53248
20 FOR T=0 TO 62
30 READ A
40 POKE 192*64+T,A
50 NEXT T
```

If you have enough data values in your program to define eight different Sprites, you must set the pointers for all eight Sprites, then read in all the data values into the proper locations. Let's follow this process in detail.

Assigning Individual Pointers

Because all eight Sprites will be different, and will have their own unique DATA values, each will need to be assigned a Sprite pointer.

```
20 POKE 2040,192
30 POKE 2041,193
40 POKE 2042,194
50 POKE 2043,195
60 POKE 2044,196
70 POKE 2045,197
80 POKE 2046,198
90 POKE 2047,199
```

Of course, this could also be accomplished with a simpler loop:

```
20 FOR T=0 TO 7
30 POKE 2040+T,192+T
40 NEXT T
```

Entering Data Values

Now that we have indicated the area in memory where each of the DATA values will be found, we must read the DATA values into the correct areas.

```
10 FOR T=0 TO 62
20 READ A
40 POKE 192*64+T,A
50 NEXT T
```

This will place the 63 data values for Sprite #0 into memory locations (192*64)+0 through (192*64)+62.

We do the same for the other seven Sprites, but change the area number.

```
10 FOR T=0 TO 62:READ A:POKE 193*64+T,A:NEXT T
20 FOR T=0 TO 62:READ A:POKE 194*64+T,A:NEXT T
30 FOR T=0 TO 62:READ A:POKE 195*64+T,A:NEXT T
40 FOR T=0 TO 62:READ A:POKE 196*64+T,A:NEXT T
50 FOR T=0 TO 62:READ A:POKE 197*64+T,A:NEXT T
60 FOR T=0 TO 62:READ A:POKE 198*64+T,A:NEXT T
70 FOR T=0 TO 62:READ A:POKE 199*64+T,A:NEXT T
```

By now, you can see the pattern forming. If we are going to read each of the eight Sprites into sequential memory areas, we can accomplish this in a simpler way:

```
10 FOR B=0 TO 7
20 FOR T=0 TO 62
30 READ A
40 POKE (192+B)*64+T,A
50 NEXT T
60 NEXT B
```

Defining Sprites From The Same Locations

But you won't always need eight different Sprites in your programs. Suppose you want eight Sprites, all with the same shape, and therefore all with the same 63 DATA values. All you have to do is give the same memory area values in each of the eight Sprite pointer registers:

```
10 POKE 2040,192
20 POKE 2041,192
30 POKE 2042,192
40 POKE 2043,192
50 POKE 2044,192
60 POKE 2045,192
70 POKE 2046,192
80 POKE 2047,192
```

This example will have all eight Sprites defined with the DATA values stored in memory bank 192.

Sequence Counts

Remember that the order of the DATA statements in your program determines how they will be read — you can't instruct the program to start reading DATA from the middle of the DATA statements. This means that if you have placed the DATA statements for Sprite #3 first in the program, you have to be sure that when you read in those first 63 values, you are reading them into the memory area locations you have designated for Sprite #3.

To put this another way, you must be able to read the Sprite DATA statements in the order your program is expecting them. If your program needs to read the DATA statements for Sprites #1, #2, #3, and #4 in that order, the DATA statements better be in that same order — first the 63 values for Sprite #1, then the 63 values for Sprite #2, then the values for #3, and lastly the values for #4.

To make a mistake of this type will not cause any errors when you run the program. After all, the computer doesn't care which Sprite ends up in which memory area. But you will end up greatly confused when you try to change the color or screen coordinates for your Sprites, and things don't turn out the way you are expecting them to.

CHAPTER 5

TURNING SPRITES ON AND OFF

To see Sprites on the screen, they must first be enabled. This means a value must be poked into a register to turn on each Sprite you wish to have displayed on the screen.

Unlike the color registers, where each Sprite had a separate register, all eight Sprites are enabled and disabled from a single register. This is possible because every register contains eight bits of information within one byte. Therefore, to turn on all eight Sprites, we must poke a value of "1" or "on" into each of the eight bits in this register.

ENABLING YOUR SPRITES

The enable register is at location V+21. The Sprite Enable Register's eight bits can be represented like this:

TURNING SPRITES ON AND OFF

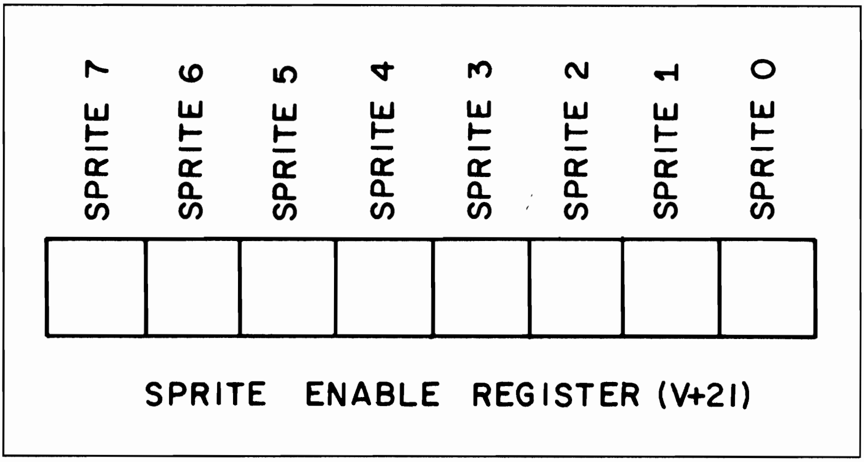


Figure 5-1

If all eight bits contain a "1" then all eight Sprites are enabled. If we want to enable selected Sprites, we translate the binary number shown in the register into a decimal number. This is the value we then poke into the register.

To enable Sprites #2, 4, and 7:

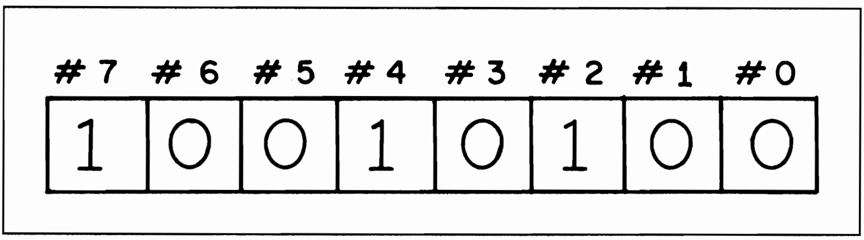


Figure 5-2

This is how we want the contents of the register to look. The decimal equivalent of this number is 148, so our POKE statement would be:

```
10 POKE V+21,148
```

To turn off all the Sprites, we simply poke all zeroes into the register at V+21.

```
POKE V+21,0
```

When To Enable Sprites Within A Program

When beginning to write Sprite programs, you must remember that as soon as you poke values into the Sprite Enable Register, those Sprites will appear on the screen, if they have valid screen coordinates within the visible screen area. (Assigning screen coordinates will be covered in the next chapter.) The program will not check to see if you have already poked valid values into the Sprite memory areas indicated. It will happily use as Sprite data any garbage which may be in those locations at the moment. What you will see on the screen will most likely be a tremendous mess.

How To Enable/Disable Selected Sprites

It is very easy to turn on or turn off a few Sprites, provided you always know what should be done with each of the eight Sprites every time.

But in many applications you want to control each Sprite independently. You might want to enable Sprite #4, for instance, but you won't necessarily know what will be going on with the other seven Sprites. So you can't just use

```
190 POKE V+21,16
```

because that will enable Sprite #4, all right, but it will also turn off all the other Sprites.

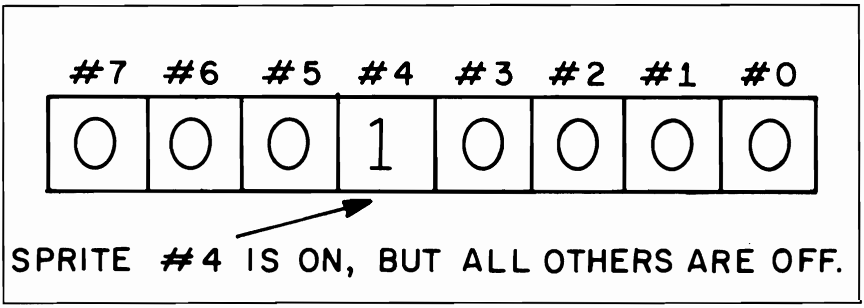


Figure 5-3

This is not what we want. We want to leave the seven other bits unaffected and only change the bit for Sprite #4.

To place a "1" in the bit for Sprite #4 and leave the other bits as they are, use the statement:

```
10 POKE V+21,PEEK(V+21) OR (2↑4)
```

If you aren't sure what this statement is doing, read about Boolean functions in Chapter 13.

The general form of this statement, to ENABLE any selected Sprite:

```
10 POKE V+21,PEEK(V+21) OR (2↑SN)
```

where SN stands for the Sprite number (#0 thru 7).

The general statement to DISABLE any selected Sprite would be:

```
10 POKE V+21,PEEK(V+21) AND (255-2↑SN)
```

where SN stands for the Sprite number (#0 thru 7).

A Sprite will stay enabled in the memory until you POKE it off, redefine it, turn off your computer, or hit RUN/STOP and RESTORE.

CHAPTER

6

POSITIONING YOUR SPRITES ON THE SCREEN

Each Sprite must have two coordinates for positioning on the television screen or monitor. You must specify the vertical positioning (Y-coordinate) and the horizontal positioning (X-coordinate). Each Sprite has two unique registers for storing these values. The POKE statement is used to place values in these registers.

POSITIONING REGISTERS

Sprite #	*X Register*	*Y Register*
0	v+0	v+1
1	v+2	v+3
2	v+4	v+5
3	v+6	v+7
4	V+8	V+9
5	V+10	V+11
6	V+12	V+13
7	V+14	V+15

POSITIONING YOUR SPRITES ON THE SCREEN

To place Sprite #0 at 50 dots over and 75 dots down on the screen, you poke these values into the registers for Sprite #0:

```
110 POKE V+0,50
120 POKE V+1,75
```

SCREEN DIMENSIONS AND POSITIONING SPRITES

For screens with 40 columns and 25 rows, this illustration shows valid values for X and Y coordinates. When you look at the picture, however, notice that there is a difference between valid values and the values for the visible viewing area.

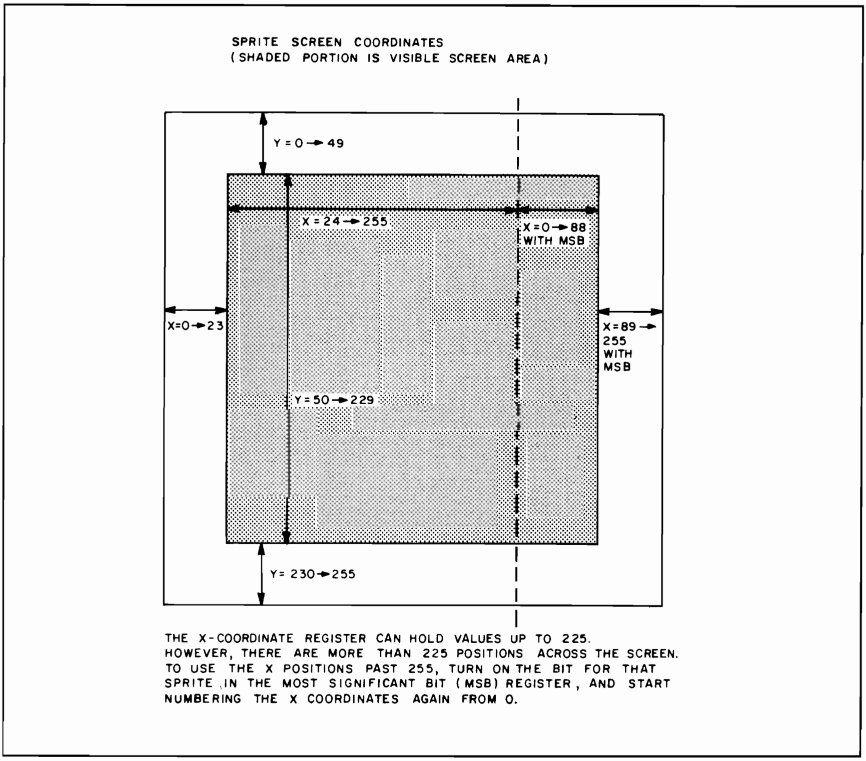


Figure 6-1

It is easiest to explain this with an example.

Positioning Example

If we have defined a Sprite and we want to position it in the upper left hand corner of the screen, we can't simply set an X-coordinate of zero and a Y-coordinate of zero. Even though those are valid values, they will position the Sprite in an area of the screen that is beyond the visible viewing area. To place the Sprite in the upper left corner, the values needed are X-coordinate of 24 and Y-coordinate of 50. By looking at the chart, you see this same idea holds for all sides of the screen.

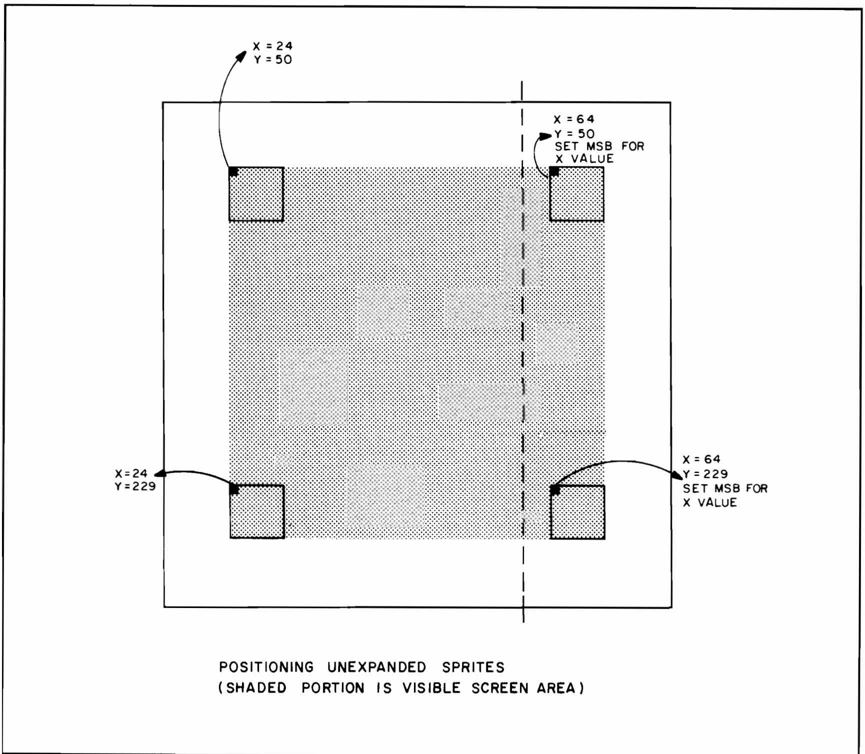


Figure 6-2

Consider The Whole Sprite!

An important thing to realize here is that when we position a Sprite, we are referencing the upper left corner of the Sprite.

POSITIONING YOUR SPRITES ON THE SCREEN

Remember, a "regular" Sprite is 24 dots wide by 21 dots long. When you set the X and Y coordinates, those coordinates are really designating where the upper left corner dot in that Sprite will be positioned.

You are always working with a 24 by 21 dot Sprite size, when you set screen positioning for normal sized Sprites. It does not matter to the computer whether you filled in every dot in the entire Sprite, or left 95% of it empty. You are still positioning the entire Sprite — all 24 columns by 21 rows. When you want to place Sprites at exact locations, or in the exact center of a screen area, this takes a bit of arithmetic. It also means that you must know exactly how your Sprite figure is placed within the 24 by 21 Sprite grid.

If your screen or monitor is of a different size than 40 wide by 25 long, consult your Commodore manual for valid X and Y coordinate changes.

MOVING A SPRITE OFF THE VISIBLE SCREEN

There will be applications where it will be very handy to be able to move a Sprite off the visible portion of the screen without disabling that Sprite. You can set the coordinates for a Sprite to any valid numeric value, even though the Sprite may not be visible when positioned at certain coordinates.

Overlapping And Expanding

As you will see in Chapter 7 about Sprite priorities, you do not have to guard against overlapping Sprites on the screen, or against assigning them the same coordinates. Neither of those conditions will cause an error. In fact, being able to overlap Sprites is a very handy feature of Sprite graphics, and one that will allow you to produce very professional-looking graphics.

In Chapter 8 you will learn how to expand Sprites along the X and Y axes. This will mean some refiguring for positioning Sprites on the screen, since they could now be twice the size of the originals. However, no matter what size the Sprite, the positioning coordinates always reference the upper left corner.

A Sprite positioning chart for expanded Sprites is included in Chapter 8.

MOST SIGNIFICANT BIT

When looking at the X-coordinate position values in the previous chart, you may have spotted a problem.

Remember that an X-coordinate register such as V+39 has only eight bits. If each of the eight bits is turned on, the resulting binary number is 11111111, or decimal 255. Therefore, the largest number value we could poke into register V+39 is 255. So — what are we going to do with X-coordinate values ranging from 256 thru 343, which is the rightmost edge of our screen?

You'll be happy to know that you do not have to leave the right third of your screen Sprite-less. There is a way to use the X-coordinate positions greater than 255. This is where the Most Significant Bit comes in.

Most Significant Bit Register

The Most Significant Bit register is located at V+16. It contains a bit for each of the eight Sprites. If the bit for a particular Sprite is turned "on," that indicates that the X-coordinate given is past the first 255 positions, and numbering has started over again at 0. Here is an example.

Most Significant Bit Example

We wish to position Sprite #2 at 100 dots down and the equivalent of 300 dots across. Setting the Y-coordinate is easy:

```
10 POKE V+5,100
```

To set the X-coordinate, we can't just say POKE V+4,300 because that will cause an "Illegal Quantity" error when you try to execute the program.

The X-coordinate for Sprite #2 will have to make use of the Most Significant Bit. We will enable the Most Significant Bit for Sprite #2, then set the X-coordinate value at (300-256), or 44.

```
10 POKE V+5,100
20 POKE V+16,4
30 POKE V+4,45
```

Line 20 has turned on the bit in the Most Significant Bit register for Sprite #2.

#7	#6	#5	#4	#3	#2	#1	#0
0	0	0	0	0	1	0	0

MOST SIGNIFICANT BIT REGISTER (V+16)

Figure 6-3

The decimal equivalent of this binary number is 4. That is the value we poked into register V+16, to show that the value given in the X-coordinate register for Sprite #2 (V+4) is beyond the first 255 dots. In the Commodore manuals, this value is referred to as the Right X Position.

(In case you are wondering why we arrived at the Right X position by subtracting 256 from 300, instead of subtracting 255 from 300, remember that the first coordinate value past 255 is zero, not 1. To make the Right X value come out correctly, we must subtract 256, not 255. If you're still not convinced, try it both ways to see for yourself.)

To set the Most Significant Bit register for each Sprite, these values are used:

SETTING THE MOST SIGNIFICANT BIT REGISTER	
Sprite #	*Most Significant Bit Value*
0	POKE V+16,1
1	POKE V+16,2
2	POKE V+16,4
3	POKE V+16,8
4	POKE V+16,16
5	POKE V+16,32
6	POKE V+16,64
7	POKE V+16,128

That sounds easy enough, right? Well, think about it for a minute.

If you are working with more than one Sprite on the screen, you may have the Most Significant Bit for several of them turned "on." Then you need to turn on the Most Significant Bit for Sprite #7, for example. You can't use the POKE v+16,128 statement for this purpose, because along with turning on the bit for Sprite #7, it will turn off the other seven bits.

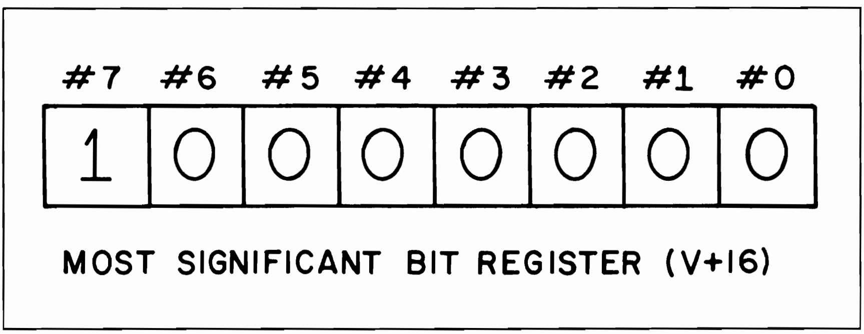


Figure 6-4

This is the end result in register V+16 if you poke it with the value 128.

Boolean And/Or Statements

So, how do you overcome this problem? You use the Boolean OR and Boolean AND to selectively turn on and off certain bits in the register without affecting the other bits. (The use of these Boolean operations is covered in Chapter 13, if you need an explanation.)

To selectively turn ON the Most Significant Bit for a Sprite, use this statement:

```
10 POKE V+16,PEEK(V+16) OR (2↑SN)
```

where SN is the Sprite number (from #0 through #7).

To selectively turn OFF the Most Significant Bit for a Sprite, use this statement:

```
10 POKE V+16,PEEK(V+16) AND (255-2↑SN)
```

where SN is the Sprite number (from #0 through 7).

CHAPTER 7

SPRITE PRIORITIES

Each Sprite has a number, ranging from zero through 7. Quite simply, Sprite priorities are set by their numbers. The lower the number, the higher the priority.

Sprite #0 will have priority over any other Sprite, because it has the lowest number. This means that if Sprites #0 and #5 pass through the same location on the screen, Sprite #0 will pass "in front of" Sprite #5. If several Sprites are overlapping, even slightly, in the same portion of the screen, each Sprite will be in front of or behind the other Sprites, based on the Sprite numbers.

If a Sprite in front has any "holes" or unshaded areas in it where the background would normally show through, another Sprite passing behind it will show through these holes. This gives a three-dimensional effect.

The background on the screen has the lowest priority of all. Sprites will usually pass in front of the background. This background can include the usual solid color, or text, or character graphics, or any other graphics modes.

No matter what other graphics modes you use in your programs, your Sprites will keep their shape and operate independently. This is a very handy feature for many applications.

SPRITE TO BACKGROUND PRIORITY REGISTER

There is a way to change the priority of the Sprites to the background, so that Sprites can actually pass BEHIND background figures. This is done by turning on a Sprite's bit in the Sprite Background Priority Register, located at register V+27.

Each Sprite has a bit in this register. If a Sprite's bit is turned "on," then the background has a higher priority than the Sprite, and that Sprite will pass BEHIND any background figure it crosses.

If a Sprite's bit is turned "off" in register V+27, then that Sprite has a higher priority than the background. When the Sprite and a background figure cross paths, the Sprite will pass in front of the background figure.

You can POKE a value into this register to turn on the Sprites you wish to have lower priority than the background figures.

If you want Sprite #4 and #6 to have lower priority than the background figures (to do this, their bits must be set to 1), this can be accomplished with a POKE statement for register V+27:

```
20 POKE V+27,80
```

Or, if you don't feel up to any arithmetic at the moment, you can let the computer do the addition, also:

```
20 POKE V+27,16+64
```

What we have done in this POKE statement is to add the decimal equivalents of the Sprites we wish to have a lower priority than the background figures.

If you have a hard time remembering whether setting a bit to 1 in this register makes the background or the Sprite have higher priority, think of it in the same way as you do the Sprite priorities to each other. The lower the number, the higher the

priority. Therefore, if a Sprite has the lowest number possible in this register, (namely zero) it will have the highest priority, and will pass in front of the background figures.

You can also use a selective POKE statement to change a Sprite's priority in relation to the background during the course of a program, if you wish to leave the Sprite-to-background priority of all the other Sprites unchanged.

To set a Sprite's bit to 1, therefore making the background figures have a higher priority:

```
10 POKE V+27,PEEK(V+27) OR (2↑SN)
```

where SN is the Sprite number from #0 through #7.

To set a Sprite's bit to 0, therefore making the Sprite have higher priority over the background figures:

```
10 POKE V+27,PEEK(V+27) AND (255-2↑SN)
```

where SN is the Sprite number from #0 through #7.

PLANNING SPRITE PRIORITIES IN PROGRAMS

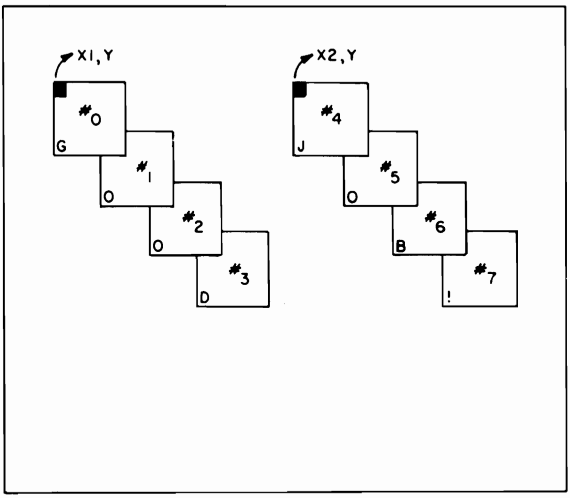
When designing screen layouts and program logic, it is easy to forget that you cannot change Sprite to Sprite priority in the middle of a program.

By this I mean that Sprite #0 is always going to have priority over (pass in front of) Sprite #5, for instance. If you don't think about this fact as you are planning what Sprites #0 and #5 have to do on the screen, it can be very time-consuming trying to juggle the Sprite memory assignments after the fact, trying to make things work out right.

As an example, look at the color illustration with the block letters spelling out the words "GOOD JOB!" When I laid out this screen, I knew I wanted the letters to appear in a certain pat-

tern. I wanted the "J" in "JOB" to overlap the "O" slightly, and I wanted the corner of the "O" to overlap the "B," and I wanted the corner of the "B" to overlap the exclamation point block.

To do this, I had to be sure that I had defined the Sprite numbers so this would be possible. Of all the letters in the word "JOB," the letter "J" had to be the lowest numbered Sprite, so it would appear in front of the other Sprites. This diagram will show you how I planned the screen. Numbering the Sprites like this is very helpful in keeping track of priorities, especially if you are using many Sprites on one screen. Along with the Sprite numbers on this diagram, you can see that I also planned the mathematical relationships between the X and Y coordinates I would use. This helped me code the program faster and with fewer errors.



X1 = 24
X2 = 135
Y = 50
A = 35

SPRITE NUMBER	X-COORDINATE	Y-COORDINATE
0	X1	Y
1	X1+A	Y+A
2	X1+2A	Y+2A
3	X1+3A	Y+3A
4	X2	Y
5	X2+A	Y+A
6	X2+2A	Y+2A
7	X2+3A	Y+3A

Figure 7-1

STRANGE AND UNEXPECTED COLOR EFFECTS

You may find, with your particular television set or monitor, that some overlapping Sprites will produce color phenomenon that you had not anticipated. Especially with small areas of color surrounded by large blank areas within a Sprite, when a background color shows through, or that Sprite overlaps another, the colors shown may change. An example of this can be seen if you look closely at the "steam" rising from the coffee cups in one of the color illustrations in this book. In each case, the steam was defined to be the same color as the cup itself — these are all single-color Sprites. Yet, in some places the dots making up the steam show up as several colors! This is just a function of how my particular monitor works. Experimentation with your TV or monitor may produce equally interesting and useful quirks.

CHAPTER

8

EXPANDING YOUR SPRITES

One register in the Video Chip gives you the ability to expand your Sprites along the X-axis, and another will do the same along the Y-axis. Each Sprite has its own bit within each of the two registers. If that bit is set to 1, that Sprite will be expanded. If the bit is zero, the Sprite will be normal size.

VERTICAL EXPANSION

#7	#6	#5	#4	#3	#2	#1	#0
0	1	0	1	1	0	0	0

VERTICAL EXPANSION REGISTER

Figure 8-1

If the vertical expansion register, located at V+23, contained the above binary value, Sprites #3, #4, and #6 would be expanded to twice their normal size in the vertical direction. The other five Sprites would be normal. To place this value in register V+23, use a POKE statement:

```
10 POKE V+23,88
```

or do it the easy way . . .

```
10 POKE V+23,8+16+64
```

HORIZONTAL EXPANSION

The horizontal expansion register is located at V+29. It works the same way. To expand Sprites #5 and #6 along the X-axis, for example, we would need to set the binary values of register V+29 to look like this:

#7	#6	#5	#4	#3	#2	#1	#0
0	1	1	0	0	0	0	0

HORIZONTAL EXPANSION REGISTER

Figure 8-2

The decimal equivalent of this binary number is 96, so the POKE statement would be:

```
130 POKE V+29,96
```

EXPANDING YOUR SPRITES

Any one Sprite or a combination of Sprites can be expanded using this method. You can expand a Sprite vertically, horizontally, or both:

```
130 POKE V+23,15 (vertical expansion)
140 POKE V+29,15 (horizontal expansion)
```

This example would expand Sprites #0, 1, 2, and 3 along both axes. They would appear on the screen twice their normal size. Keep in mind that the resolution of the expanded Sprites does not change — they are just made up of more pixels on the screen.

In one of the color illustrations in this book, examples are shown of Sprites expanded vertically only, horizontally only, in normal size, and expanded in both directions. Sprite expansion is useful when you need larger versions of a Sprite, or want to show action by expanding a Sprite, or wish to call attention to a particular Sprite on the screen.

HOW SPRITES EXPAND

An expanded Sprite ends up with twice the length and width it had in normal size. A normal Sprite is 24 dots wide. Therefore, expanding a Sprite horizontally results in a Sprite that is 48 dots wide.

Expanding a Sprite vertically changes the size of the Sprite from 21 dots long to 42 dots long. The number of DATA statements needed to define that Sprite does not change — the Sprite is simply displayed larger on the screen. Setting the color of the Sprite is done the same, and enabling the Sprite is handled exactly as if the Sprite were normal size.

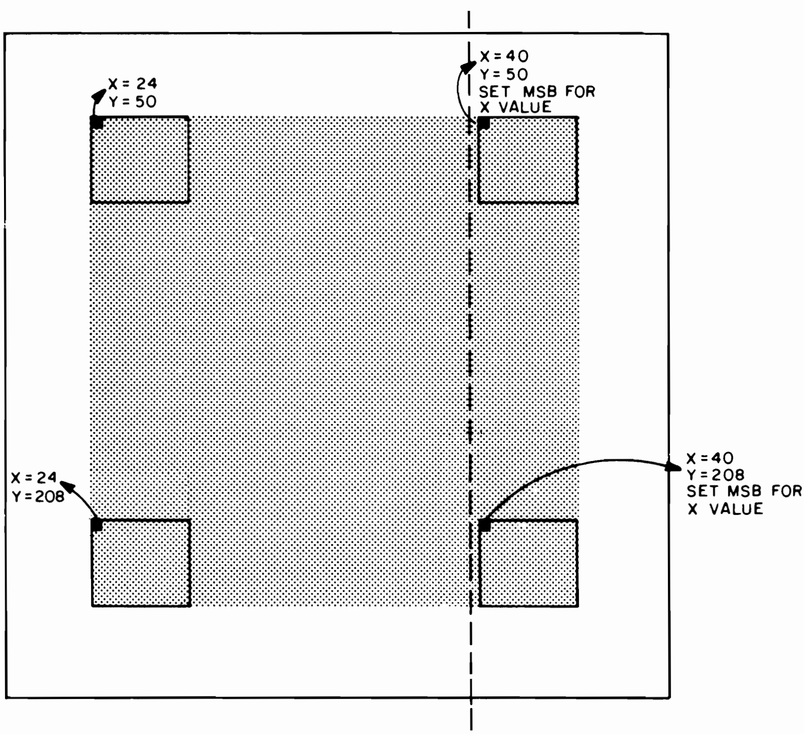
Positioning Expanded Sprites

You must allow for the expansion when you choose the X and Y coordinates to position an expanded Sprite on the screen. If the Sprite has been expanded in both directions, you are now dealing with a 48 by 42 dot Sprite, instead of a 24 by

21 size. The screen coordinates you give will still reference the top left corner dot in the Sprite. But since the Sprite is bigger, it will take up more room on the screen.

Having the expansion feature available is very useful when you wish to make fairly detailed Sprites. Small details will show up better when the Sprite is a larger size, of course. However, for arcade-style games where you are trying to pack a lot of action onto a small screen, the expanded Sprite may be more of a hindrance. It all depends on your application.

This chart will help you see how much more room you must allow for your expanded Sprites.



POSITIONING EXPANDED SPRITES
(SHADED PORTION IS VISIBLE SCREEN AREA)

Figure 8-3

Expanding Selected Sprites

As with setting our other registers, sometimes you will want to be able to expand or "unexpand" selected Sprites without affecting the bit settings in the Sprite expansion registers for the remainder of the Sprites.

If you have expanded all eight Sprites in the horizontal direction, and you wish to return Sprite #0 back to normal size, you can't use POKE V+29,0 because while that will return Sprite #0 to normal size, it will also return all the other Sprites to normal size as well.

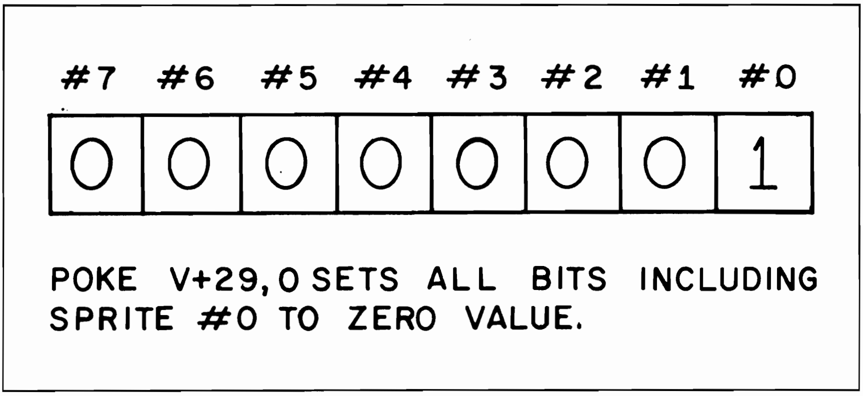


Figure 8-4

To selectively turn on or off bits in the Sprite expansion registers, use these statements:

To expand selected Sprites in the horizontal direction:

```
10 POKE V+23,PEEK(V+23) OR 2↑SN
```

where SN is the Sprite number (from #0 through #7).

To expand selected Sprites in the vertical direction, use the same statement format as above, but change the register location to V+29:

```
10 POKE V+29,PEEK(V+29) OR 2↑SN
```

To UNEXPAND selected Sprites in the horizontal direction:

```
10 POKE V+23,PEEK(V+23) AND (255-2↑SN)
```

where SN is the Sprite number (from #0 through #7).

To UNEXPAND selected Sprites in the vertical direction, use the same statement format as above, but change the register location to V+29:

```
10 POKE V+29,PEEK(V+29) AND (255-2↑SN)
```

A SAD LAST NOTE

Unfortunately, if you design an ugly Sprite in normal size mode, it doesn't get any better looking when you expand it to twice its size. For those of us with no artistic talent, this is one of those cruel facts of life.

CHAPTER 9

MOVING AND ANIMATING SPRITES ON THE SCREEN

Movement and animation is one of the capabilities of your Commodore 64 graphics that can make your displays most exciting. Luckily, moving Sprites on the screen is as simple as changing the X or Y coordinates.

MOVING SPRITES VERTICALLY

Let's define Sprite #0 on the screen, then move it straight downward. This Sprite is normal sized, so at least some of it will be visible on the screen if we keep the Y-coordinates within the range of 30 and 249. We'll set the X-coordinate at 100.

```
10 POKE V+0,100
20 FOR Y=30 TO 249
30 POKE V+1,Y
40 NEXT Y
```

In using a FOR-NEXT loop, the default value for the increment amount is +1. A movement of one dot down the screen at a

time will produce very smooth motion. But if we wanted something else, we could code it this way:

```
10 POKE V+0,100
20 FOR Y=30 TO 249 STEP 3
30 POKE V+1,Y
40 NEXT Y
```

Adding the STEP 3 to statement 20 increases the increment amount to 3 instead of 1. Now the Y-coordinate value will jump by threes instead of one dot at a time. The result will be a faster, less smooth movement down the screen.

MOVING SPRITES HORIZONTALLY

Horizontal movement is just as easy, until you reach the right hand portion of the screen, where the X value has reached 255 and you can't go any farther until you set the proper bit in the Most Significant Bit register. (Consult Chapter 6 if you need help on what the Most Significant Bit register does.)

As you recall, the X values can run from 1 through 343 and still have some portion of an unexpanded Sprite visible on the screen. However, 343 is not a valid value for the X coordinate. (Numbers greater than 255 will not fit in the 8-bit X-coordinate register.) Therefore, we need to set up a simple routine that will turn on the proper bit in the Most Significant Bit register when the X value exceeds 255.

For this example we will use a Y value of 100. We want the X values to range from 1 through 255, then set the Most Significant Bit for Sprite #0 and continue across the screen with X values 0 through 87 (87 = 343 − 256).

```
10 POKE V+1,100
20 FOR X=1 TO 343
30 IF X>255 THEN POKE V+16,1:
   POKE V+0,X-256:GOTO 50
40 POKE V+0,X
50 NEXT X
```

The above example will work, but it can be expressed in a more useful way.

```
10 POKE V+1,100
20 FOR X=1 TO 343
30 RX=INT(X/256)
40 LX=X-(256*RX)
50 POKE V+16,RX
60 POKE V+0,X
70 NEXT X
```

Program Highlights Here is how this program facilitates horizontal movement.

Line 10 sets the vertical coordinate to 100.

Lines 20 and 70 show we will step the X values through the range of 1 to 343, using the default increment of 1.

Line 30 divides the X value by 256, and then takes the integer value of the quotient. This results in either a zero, if the X value is less than 255, or "1" if the value is between 256 and 343.

Line 40 computes the X value to be used. The expression (256*RX) will be a zero if the X value was less than 256. Therefore, the X value will not be changed.

Line 50 sets the Most Significant Bit register to one or zero.

This will turn on or off the bit for Sprite #1.

Line 60 pokes the X value for Sprite #0.

To see more examples of moving Sprites, look carefully at the sample programs in Chapter 12.

DIAGONAL MOVEMENT

Sprites can be moved diagonally by changing both the X and Y coordinates at the same time. The angle of the diagonal will depend on how fast the two coordinates change in relation to one another.

The easiest way to figure how many dots in the X direction you want to move for so many dots in the Y direction is to count the total X dots from the starting point to the ending point, then

how many Y dots from starting to ending points, and divide to get a ratio:

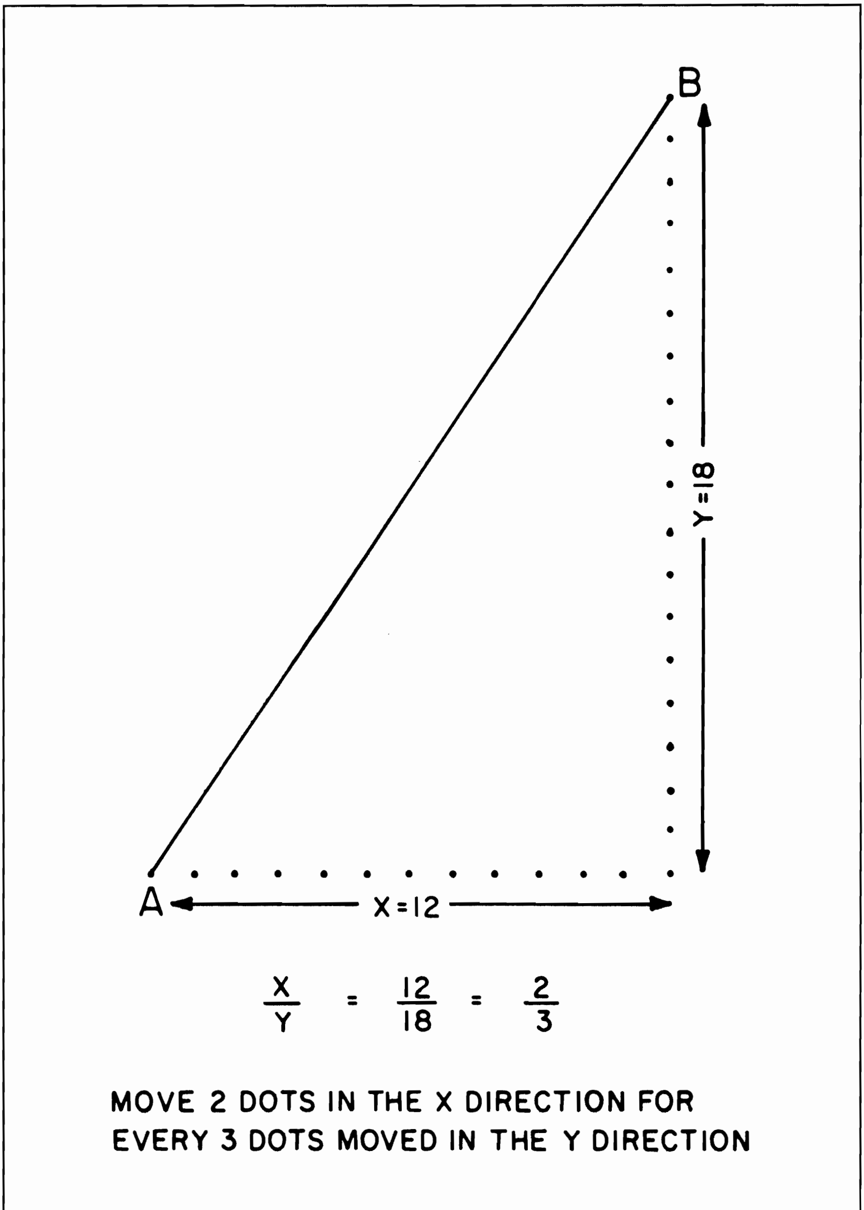


Figure 9-1

If we want to move from point A to point B on this diagram, we want to go a total of 12 dots in the horizontal direction for every 18 dots we move in the vertical direction. Dividing this out to the lowest fraction, we want to move 2 dots in the X direction every time we move 3 dots in the Y direction. Now, you're probably thinking that not every ratio will divide out that evenly. You're right. BUT, you cannot move a Sprite a fraction of a dot in either direction, so you're going to have to come out with numbers you can readily reduce, no matter how you look at it.

Plus, if you have come up with some weird ratio such as moving 17 X dots for every 11 Y dots, your movement on the screen is going to be pretty jerky! So the moral of this lecture is to simplify your diagonal movement so you can move a reasonably small number of dots at a time in either direction, if you want to achieve smooth diagonal movement.

ANIMATING SPRITE FIGURES

When animators create cartoons, they draw many versions of the same figure going through the stages of a motion. Then these slightly different images are projected one after another in the same spot, and the speedy changing of one drawing for another tricks our eyes into thinking we are seeing smooth motion.

Sprite graphics works in the same fashion. Our task is to rapidly display different versions of the same Sprite. As an example, we'll use a Sprite shaped like a bird in flight. We will need five different views of this bird, each with the wings in a slightly different position. Let's logically consider the different ways this could be accomplished.

As you have found out when you executed your first Sprite programs, it takes time for the computer to read all 63 DATA values for each Sprite into memory. This is by far the most time-consuming portion of your program's execution.

A Fast-Flying Fowl

If we need to have our bird shape change rapidly through the five versions of wing positions, we can't afford the time needed to define Sprite #0 as bird picture A, then read in all new data statements to change that Sprite's DATA values so it will look like bird B. This method would work, but it takes far too long.

Instead, we can store all five versions of the bird in the memory at the beginning of the program. Then, when we want to animate the bird picture, we will define a single Sprite to point to the memory area where we have stored bird picture A. When we want to display bird picture B, we will just switch the pointer value for that same Sprite to point to the memory area where we have bird B stored. Using this method, we can switch rapidly through all versions of the bird, yet still keep the same Sprite enabled.

Using this method, we are able to animate all eight Sprites, if we wish. Our only limitation is how much room we can squeeze out of the available memory, to contain different versions of all eight Sprites. Using the suggested memory areas in Chapter 4, we can easily have 64 different Sprite versions available. If you need more than this, you will have to find more room in memory.

Effective Memory Use

But remember, the same memory area in storage can be pointed to by as many Sprite pointers as you wish. In other words, if your entire program consists of spaceship Sprites zooming around on the screen, perhaps you will only need four different versions of that spaceship to fill all your needs. At any one time, any number of Sprites could be pointing to a single version of that spaceship in memory. In this way, you can get maximum mileage out of only four sets of Sprite DATA statements.

Another Animation Technique

An alternative animation technique is to define each of the eight Sprites as a version of the same animated figure. Then all eight Sprites should have their X and Y coordinates set at the same screen positions. To accomplish the animation, enable and disable each Sprite in the desired sequence. Don't forget to turn on and off the multi-color bits if one of the Sprites in the sequence is multi-colored (more about this in Chapter 11).

Extremely Fast Animation

If you need extremely fast animation, you will need to go to machine language. Using BASIC, it is not possible to move and animate Sprites at the same time and achieve any great speed. However, for most applications, BASIC and Sprite graphics will be more than adequate for your needs.

Chapter 12 contains some excellent examples of different ways to animate Sprites. It will be worth your while to study the listings carefully.

CHAPTER 10

COLLISION DETECTION

Two types of collisions can be automatically detected when you program with Sprites. The first type is a collision between two Sprites and the second is a collision between a Sprite and the background. Sprite collision detection is important because it is often the most important element in computer game designs.

SPRITE-TO-SPRITE COLLISIONS

How can you tell if two Sprites have collided? Register V+30 contains a bit for each of the eight Sprites. If a Sprite's bit is a 1, then that Sprite is touching some other Sprite.

In the following example of the contents of register V+30, bits for Sprites #7 and #5 are turned on. This shows that some non-zero part of Sprite #5 is touching some non-zero part of Sprite #7.

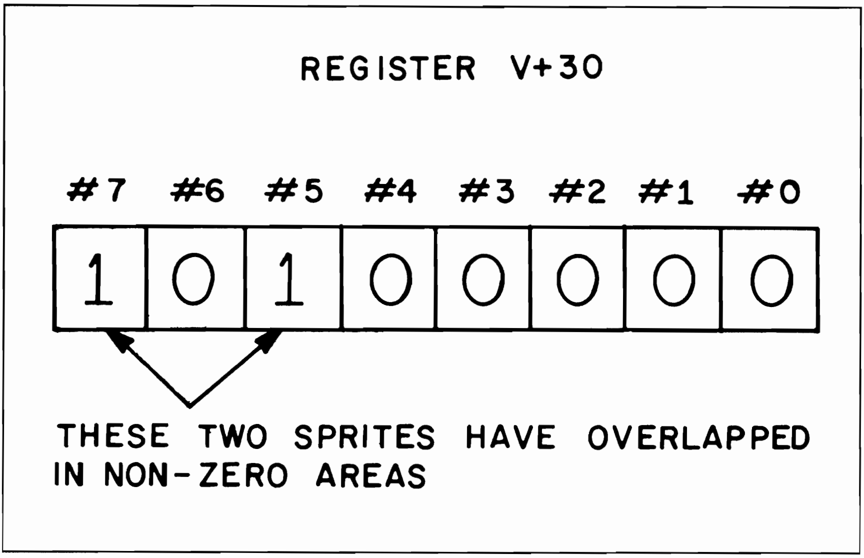


Figure 10-1

Peeking At Sprites

Now the question is, how do you get "inside" the register to see which bits have been turned on?

To do this, we use a PEEK statement as part of an IF-THEN.

```
210 IF PEEK(V+30) AND 160 = 160 THEN {take some action}
```

This statement accomplishes two things. First, it PEEKs at the value in register V+30. Then it uses that value in a Boolean AND operation with the value 160. If the results of the Boolean AND equal the value 160, then the statement is true, and the action following the THEN part of the statement is carried out.

If you are not sure how this Boolean AND operates, it is discussed in detail in Chapter 13.

It is important to realize that due to the way the PEEK statement works, once you have PEEKed into a register, all those

bits will be reset to "off" or zero. Therefore, if you need to keep track of which bits were on, you must save this information in memory somewhere so you can access it later.

Detecting Collisions

This same statement format can be used to detect collision involvement for any Sprite(s) you like. The general format of the statement is:

```
210 IF PEEK(V+30) AND X = X THEN {take some action}
```

where X is the total decimal value of the Sprite bit(s) you want examined for collision. So, to look for a collision involving Sprite #4, you know that the bit for Sprite #4 in the register would have to be turned on. The decimal equivalent of this value is 16. That is the value you use for X in the PEEK statement looking at the contents of register V + 30.

If Sprite #4 is involved in a collision, you know its bit must be a "1", like this:

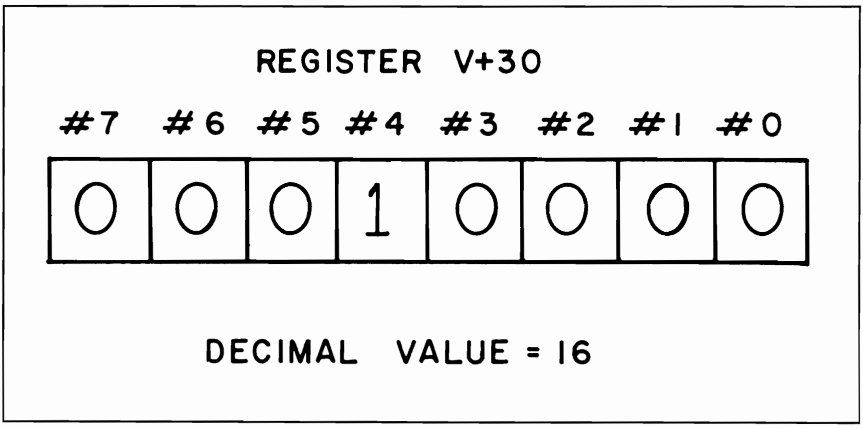


Figure 10-2

(Actually, another Sprite's bit would have to be on as well — Sprite #4 can't collide with itself! But for our discussion, we are

only interested in determining if Sprite #4 is involved. We don't care at this point which other Sprite it collided with.)

Therefore, your PEEK statement is looking to see if this bit is "on." The decimal equivalent of this value is 16, so the PEEK statement would be:

```
210 IF PEEK(V+30) AND 16 = 16 THEN PRINT
"COLLISION--SPRITE #4"
```

You can check any combination of bits you wish by adding up their decimal equivalents and using that number in the above PEEK statement.

COLLISIONS WITH THE BACKGROUND

Checking for Sprite collisions with the background works the same way, except we look at register V+31.

For collision purposes, the background can consist of character graphics, text, hi-resolution graphics, or anything else except the basic background color and the background border. If any part of a Sprite is touching any part of the background on the screen, that Sprite's bit will be turned on in the V+31 register. The only exception is multi-color 1 background graphics. For purposes of collision detection, they are all zeros, and will not trigger a collison. This is handy to know if you wish to have some background elements not cause collisions if they are crossed by a Sprite.

COLLISIONS OFF THE VISIBLE SCREEN AREA

It is important to realize that Sprites can be defined in areas of the screen that are not visible to the user. However, collisions occurring in these "nonvisible" areas are still valid collisions, and they will affect the bit settings in the collision registers.

CHAPTER 11

MULTI-COLOR MODE

Up until now, all our Sprites have been defined in single-color mode. But Sprites can also be done in up to four colors, using multi-color mode. In addition to the Sprite color and the transparent "background color" you can choose from when coding a single-color Sprite, multi-color mode has two more choices: multi-color 1 and multi-color 2. Multi-color 1 is set by poking a color code into register V+37. Notice that the multi-color 1 applies to all eight Sprites. If you have more than one multi-color Sprite on the screen at a time, they will all have the same multi-color 1 value. Multi-color 2 is set by poking a color code into register V+38. It has the same limitation as multi-color 1 does — the multi-color 2 color is the same for all Sprites on the screen.

DESIGNATING COLORS IN MULTI-COLOR

Since we must have some way to indicate which of these four color choices we will be using for each dot in a multi-color Sprite, we must sacrifice horizontal resolution to accomplish this task.

MULTI-COLOR MODE

In a single color Sprite, we can individually shade any of the 24 dots on each row of the Sprite grid. But in multi-color Sprites, we will be using PAIRS of dots in each row to designate whether these pairs should be colored, and which of the four color choices they should be.

In this example, we are looking at the top three rows of a multi-color Sprite. The diagram key shows how we want to have each pair of dots colored.

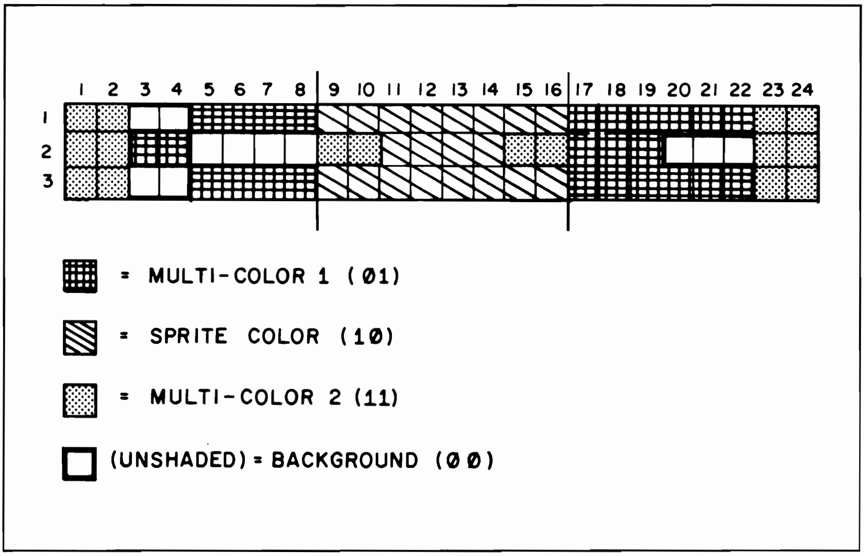


Figure 11-1

Now, each PAIR of colored dots must be coded like this:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	1	1	0	0	0	1	0	1	1	0	1	0	1	0	1	0	0	1	0	1	0	1	1	1
2	1	1	0	1	0	0	0	0	1	1	1	0	1	0	1	1	0	1	0	0	0	0	1	1
3	1	1	0	0	0	1	0	1	1	0	1	0	1	0	1	0	0	1	0	1	0	1	1	1

MULTI-COLOR 1 = (01)
SPRITE COLOR = (10)
MULTI-COLOR 2 = (11)
BACKGROUND = (00)

Figure 11-2

Converting the binary numbers to decimal equivalents, our DATA statements for the first three rows of this multi-colored Sprite would be:

```
1000 DATA 197,170,87
1010 DATA 208,235,67
1020 DATA 197,170,87
```

DISPLAYING MULTI-COLOR SPRITES

Now that we have the coded values for this portion of a Sprite, how do we get these four different colors (actually, three colors and a background color) to show up on the screen?

The background color is set when you poke the color code into location 53281. Multi-color 1 is set by poking a color code into register V+37. The Sprite color is set by using the color register for each individual Sprite, as is done with single color Sprites. The multi-color 2 register is V+38. It is set the same way as multi-color register 1.

ENABLING MULTI-COLOR SPRITES

Before a Sprite will appear on the screen in all its multi-colored glory, you must poke a value into one more register.

Register V+28 is for turning on and off multi-color mode. It has eight bits, one for each of the eight Sprites. If you want a Sprite to be multi-colored, you must poke a value into register V+28 so the bit for that Sprite will be a 1.

For example, to make Sprites #1, 2, and 3 multi-colored, the contents of register V+28 must look like this:

#7	#6	#5	#4	#3	#2	#1	#0
0	0	0	0	1	1	1	0

MULTI-COLOR MODE REGISTER (V+37)

Figure 11-3

The decimal equivalent of this value is 14, so the POKE statement would be:

```
140 POKE V+28,14
```

Realize that if you design a Sprite as multi-colored, you cannot simply turn off the bit for that Sprite in the V+28 register, and expect to see the Sprite intact, but in single color mode. You'll see a Sprite, all right, but it won't be what you expected. Most likely, it will be a royal mess.

This only makes sense. The very way we encode a Sprite as single or multi-colored makes a big difference in how the values in the DATA statements will turn out. Just turning on or off a bit in the multi-color mode register will not make any

changes to the number values stored in memory to define that Sprite.

Otherwise, multi-colored Sprites can be positioned, expanded, and generally manipulated the same as single color Sprites.

MULTI-COLOR CONSIDERATIONS

It is fairly obvious that multi-color Sprite mode makes it quite difficult to design a very complex Sprite on the grid. It is tough enough squeezing a complex shape into a 24 by 21 grid for a normal sized Sprite. Multi-color mode cuts your horizontal resolution in half, so you are essentially working with a 12 by 21 dot grid, even though each horizontal dot is twice as wide. But if your Sprite shape will lend itself readily to these constraints, multi-color Sprites can produce very striking results in your graphics programs.

SAMPLE MULTI-COLORED SPRITES

Here are the number values for three multi-color Sprites, to get you started.

Multi-color Butterfly

This butterfly uses three colors: the Sprite color, multi-color 1, and multi-color 2. It does not make use of the background color because there are no "holes" within the Sprite grid that have not been colored in. Looking at the color illustration featuring butterflies, you can see that this multi-color butterfly is not as detailed as his single-colored neighbors. Half the horizontal resolution has been sacrificed for the sake of extra colors.

```
9999 REM MULTICOLORED BUTTERFLY
10000 DATA 8
10010 DATA 0,128,10,2,128
10020 DATA 2,138,0,64,136
10030 DATA 5,80,168,21,84
10040 DATA 32,93,117,33,125
10050 DATA 125,101,253,127,103
```

```
10060 DATA 253,123,103,173,123
10070 DATA 103,173,123,103,173
10080 DATA 123,103,173,123,103
10090 DATA 173,123,103,173,123
10100 DATA 103,173,123,103,173
10110 DATA 127,103,253,95,101
10120 DATA 253,21,97,117,5
10130 DATA 32,84
```

Multi-color Sailboat

This sailboat Sprite makes effective use of all four colors available, including the background color, which is allowed to show through to form part of the boat's hull.

```
10815 REM MULTI-COLOR BOAT
10820 DATA 0,8,0
10830 DATA 0,56,0,0,248
10840 DATA 0,3,248,0,15
10850 DATA 248,0,63,248,0
10860 DATA 255,248,0,0,8
10870 DATA 0,0,8,0,0
10880 DATA 8,0,0,8,0
10890 DATA 85,89,84,85,85
10900 DATA 85,255,255,255,255
10910 DATA 255,252,255,255,240
10920 DATA 252,0,0,252,0
10930 DATA 0,240,0,0,192
10940 DATA 0,0,0,0,0
```

Multi-color Target Sprite

This simple multi-color Sprite resembles a bull's-eye target. It uses only three colors, since no background color has been allowed to peep through the body of the Sprite.

```
30950 REM MULTI-COLOR TARGET SPRITE
30960 DATA 170,170
30970 DATA 170,170,170,170,149
30980 DATA 85,86,149,85,86
30990 DATA 159,225,246,159,255
31000 DATA 246,158,179,182,158
31010 DATA 170,182,158,150,182
31020 DATA 158,150,182,158,150
31030 DATA 182,158,150,182,158
31040 DATA 150,182,158,170,182
31050 DATA 158,170,182,159,255
31060 DATA 246,159,255,246,149
31070 DATA 85,86,149,85,86
31080 DATA 170,170,170,170,170
31090 DATA 170
```

CHAPTER 12

INCORPORATING SPRITES INTO YOUR PROGRAMS

At this point, examples of Sprites used in actual programs will be more helpful than more explanations.

Each program in this chapter is written in "uncrunched" style, so they are easily read and understood. When you adapt them for your own, they will run more efficiently if you remove all the "extras" that are not needed. Program "crunching" is explained in your *Commodore 64 Programmer's Reference Guide*, if you need help.

Moving Sprites Vertically on the Screen

We'll begin with a simple program to move a single Sprite from the top of the screen to the bottom.

```
10 REM **VERTICAL MOVEMENT
15 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281,7 :POKE 53280, 5
90 POKE 2040,192
110 FOR M=0 TO 62
120 READ A
```

```
130 POKE 192*64+M,A
140 NEXT M
160 POKE V+39,6
165 POKE V+0,100
170 FOR Y = 1 TO 255
180 POKE V+1,Y
185 POKE V+21,1
190 NEXT Y
195 GOTO 170
9999 REM **SOLID BUTTERFLY
10000 DATA 3,0,192,57,129
10010 DATA 156,124,195,62,254
10020 DATA 102,127,255,36,255
10030 DATA 255,153,255,255,219
10040 DATA 255,255,255,255,255
10050 DATA 255,255,255,255,255
10060 DATA 255,255,255,255,255
10070 DATA 255,255,255,255,255
10080 DATA 255,255,255,255,255
10090 DATA 255,255,255,255,255
10100 DATA 255,127,219,254,63
10110 DATA 153,252,31,24,248
10120 DATA 14,24,112
```

Line Number	Description
15	Initializes the address of the video chip, for future reference
30	Clears the screen, and disables any stray Sprites left on the screen
40	Sets the background and border screen colors
90	Sets the Sprite pointer for Sprite #0 to memory area 192
110-140	Reads the data values and pokes them into memory
160	Sets the color for Sprite #0
165	Sets the X-coordinate for Sprite #0 at 100
170-190	Steps the Y-coordinate for Sprite #0 through the values 1 to 255

Moving Sprites Horizontally Across the Screen

This program makes it easy to see how the Right X and Left X values are used, along with the Most Significant Bit Register for switching between them.

```
10 REM **HORIZONTAL MOVEMENT
15 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281, 1:POKE 53280, 2
90 POKE 2040,192
110 FOR M=0 TO 62
120 READ A
130 POKE 192*64+M,A
140 NEXT M
160 POKE V+39,5
170 POKE V+1,150
175 POKE V+16,0
180 FOR X=1 TO 344
190 IF X > 255 THEN GOSUB 500:GOTO 300
200 POKE V+0,X
210 POKE V+21,1
300 NEXT X
310 END
499 REM SUBROUTINE FOLLOWS***********
500 POKE V+16,1
510 POKE V+0,(X-256)
520 RETURN
9999 REM **SPRITE DATA VALUES START HERE
10000 REM COFFEE CUP ****************
10020 DATA 8,66,0,4,33,0
10030 DATA 2,16,128,4,33
10040 DATA 0,8,66,0,0
10050 DATA 0,0,255,255,192
10060 DATA 255,255,192,245,85
10070 DATA 252,245,85,254,245
10080 DATA 85,199,245,85,195
10090 DATA 245,85,195,245,85
10100 DATA 195,245,85,199,245
10110 DATA 85,206,245,85,252
10120 DATA 245,85,248,245,85
10130 DATA 240,245,85,192,255,255,192
```

Line Number	Description
15	Initializes the beginning address of the video chip
30	Clears the screen, and disables any stray Sprites left on the screen
40	Sets the screen background and border colors
90	Points Sprite #0 to memory area 192
110-140	Reads the data values and pokes them into memory
160	Sets the color for Sprite #0
170	Sets the Y-value for Sprite #0 to 150
175	Turns off all bits in the Most Significant Bit Register
180-300	Steps the X-values for Sprite #0 through values 1 to 344
190	Directs the program to a subroutine if the X-value is larger than 255
500-520	This subroutine sets the Most Significant Bit for Sprite #0, and starts the X-value over again with zero (X-256)

Diagonal Movement Across the Screen

This program shows diagonal movement in all possible directions across the screen. It also includes an example of diagonal movement where the X distance traveled is not the same as the Y distance traveled.

```
10 REM **DIAGONAL MOVEMENT
15 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281,7 :POKE 53280, 5
90 POKE 2040,192
110 FOR M=0 TO 62
120 READ A
130 POKE 192*64+M,A
140 NEXT M
160 POKE V+39,6
170 FOR T = 1 TO 255
175 POKE V+0,T
```

```
180 POKE V+1,T
185 POKE V+21,1
190 NEXT T
195 FOR T = 255 TO 1 STEP -1
200 POKE V+0,T
210 POKE V+1,T
230 NEXT T
240 FOR T = 1 TO 255
250 POKE V+0,T
260 POKE V+1,(255-T)
280 NEXT T
290 FOR T = 255 TO 1 STEP -1
300 POKE V+0,T
310 POKE V+1,(255-T)
320 NEXT T
330 T1=1
340 T2=1
350 POKE V+0,T1
360 POKE V+1,T2
370 T1=T1+1:IF T1>255 THEN T1=1
380 T2=T2+2:IF T2>255 THEN T2=1
390 GOTO 350
9999 REM **MAN IN SWIM SUIT
10000 DATA 0,62,0,0,42
10010 DATA 0,0,62,0,0
10020 DATA 28,0,0,127,0
10030 DATA 1,255,192,3,62
10040 DATA 96,6,62,48,28
10050 DATA 54,56,8,62,16
10060 DATA 0,0,0,0,62
10070 DATA 0,0,62,0,0
10080 DATA 54,0,0,0,0
10090 DATA 0,54,0,0,54
10100 DATA 0,0,54,0,0
10110 DATA 54,0,0,54,0
10120 DATA 1,247,128
```

Line Number	**Description**
15	Initializes the address of the video chip, for later reference

30	Clears the screen, and disables any stray Sprites left on the screen
40	Sets the screen background and border colors
90	Sets the Sprite pointer for Sprite #0 to memory area 192
110-140	Reads the data values and pokes them into the memory locations
160	Sets the color for Sprite #0
170-190	Increments the X and Y values to move the Sprite from the top left corner of the screen to lower right corner of the screen
195-230	Changes the X and Y values by a negative increment, to move the Sprite from lower right to upper left
240-280	Moves the Sprite from lower left to upper right
290-320	Moves the Sprite from upper right to lower left
330-390	This group of statements moves the Sprite 1 dot in the X direction for every 2 dots it moves in the Y direction

Random Positioning of Sprites

The Commodore 64's random number generator can be used to choose random X and Y coordinates for Sprites.

```
10 REM **RANDOM POSITIONING OF SPRITES
15 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281,15:POKE 53280,11
90 POKE 2040,192
92 POKE 2041,192
94 POKE 2042,192
96 POKE 2043,192
110 FOR M=0 TO 62
120 READ A
130 POKE 192*64+M,A
140 NEXT M
160 POKE V+39,6
165 POKE V+40,5
170 POKE V+41,4
180 POKE V+42,8
```

```
182 POKE V+21,15
185 GOSUB 1000
190 POKE V+0,X
195 POKE V+1,Y
200 GOSUB 1000
210 POKE V+2,X
220 POKE V+3,Y
230 GOSUB 1000
240 POKE V+4,X
250 POKE V+5,Y
260 GOSUB 1000
270 POKE V+6,X
280 POKE V+7,Y
290 GOTO 185
999 REM SUBROUTINE FOLLOWS *********
1000 X=INT(RND(1)*255)+1
1010 Y=INT(RND(1)*255)+1
1015 FOR D=1 TO 50:NEXT D
1020 RETURN
9999 REM **NOTCHED WING BUTTERFLY
10000 DATA 3,0,192,57,129
10010 DATA 156,124,195,62,254
10020 DATA 102,127,255,36,255
10030 DATA 255,153,255,127,219
10040 DATA 254, 63,219,252, 31
10050 DATA 219,248, 15,219,240
10060 DATA  15,219,240, 31,219
10070 DATA 248, 63,219,252,127
10080 DATA 219,254,255,219,255
10090 DATA 255,219,255,255,219
10100 DATA 255,127,219,254,63
10110 DATA 153,252,31,24,248
10120 DATA 14,24,112
```

Line Number	**Description**
15	Initializes the address of the video chip, for future reference
30	Clears the screen, and disables any stray Sprites left on the screen
40	Sets the background and border screen colors

90-96	Sets the Sprite pointer for Sprites 0, 1, 2 and 3 to memory area 192
110-140	Reads the data values and pokes them into memory
160-180	Sets Sprite colors
182	Enables the Sprites
185-280	Goes to a subroutine to get random values for X and Y, then pokes those values for each Sprite
1000-1010	Assigns random values to X and Y in the range of 1 to 255
1015	A delay loop, which leaves the enabled Sprites on the screen long enough for you to see them

Experimentation with X and Y Coordinates

This program allows you to enter your own X and Y values, and see where the expanded Sprite ends up on the screen. If you wish to play with an unexpanded Sprite, change line 70 to poke zeroes into both registers.

```
10 REM ** EXPERIMENT WITH X AND Y VALUES
15 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281,1  :POKE 53280,1
90 POKE 2040,192
100 POKE V+28,1
110 POKE V+37,2:POKE V+38,5
120 FOR M = 0 TO 62
125 READ A
130 POKE 192*64+M,A
140 NEXT M
160 POKE V+39,0
170 POKE V+23,1:POKE V+29,1
180 POKE V+0,150:POKE V+1,150
190 POKE V+21,1
195 PRINT CHR$(147)
200 INPUT "X COORDINATE FROM 0 TO 343";X
210 IF X>343 THEN PRINT "INVALID VALUE":
    GOTO 200
220 I=INT(X/256)
```

```
230 IF I=0 THEN POKE V+16,0:POKE V+0,X
240 IF I=1 THEN POKE V+16,1:POKE V+0,
    (X-256)
250 INPUT "Y COORDINATE FROM 0 TO 255";Y
260 IF Y>255 THEN PRINT "INVALID VALUE":
    GOTO 250
270 POKE V+1,Y
280 GOTO 195
10545 REM SQUARE MAN WITH FEET
10550 DATA 255,255,255,192
10560 DATA 0,3,192,0,3
10570 DATA 192,0,3,202,138
10580 DATA 131,192,0,3,192
10590 DATA 32,3,192,168,3
10600 DATA 192,0,3,192,0
10610 DATA 3,196,0,19,197
10620 DATA 85,83,196,0,19
10630 DATA 197,85,83,192,0
10640 DATA 3,255,255,255,0
10650 DATA 195,0,0,195,0
10660 DATA 84,195,21,255,195
10670 DATA 255,85,65,85
```

Line Number	Description
15	Initializes the address of the video chip, for future reference
30	Clears the screen, and disables any stray Sprites on the screen
40	Sets the background and border screen colors
90	Sets the Sprite pointer for Sprite #0 to memory area 192
100	Defines Sprite #0 as multi-colored
110	Sets the color codes for multi-color 1 and multi-color 2
120-140	Reads the data values and pokes them into memory
160	Sets the color for Sprite #0
170	Expands Sprite #0 in both directions
180	Gives the initial X and Y coordinates for Sprite #0
190	Enables Sprite #0

200-240 Asks for your input for a valid X coordinate value, and pokes that value into the X coordinate register for Sprite #0

220 Determines if this X value will be a Right X value by dividing the X value by 256. If the integer value of the quotient is zero, the number must be 255 or less, so it will be an X position on the left portion of the screen.

250 If the integer value of the quotient is a 1, then the number you entered must be larger than 255, and we will need to set the Most Significant Bit for Sprite #0, and start our X coordinate values over at zero.

250-270 The Y coordinate value is entered in the same fashion. This time, however, we do not have to worry about values larger than 255, so the process is simpler.

Changing Sprite Colors

This program uses random numbers to choose the color codes for the Sprites.

```
10 REM **CHANGING SPRITE COLORS
15 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281, 3:POKE 53280,6
90 POKE 2040,192
92 POKE 2041,192
94 POKE 2042,192
96 POKE 2043,192
97 POKE 2044,192
98 POKE 2045,192
99 POKE 2046,192
100 POKE 2047,192
110 FOR M=0 TO 62
120 READ A
130 POKE 192*64+M,A
140 NEXT M
150 FOR YC=1 TO 15 STEP 2
160 Y=INT(RND(1)*175)+50
162 POKE V+YC,Y
```

```
165 NEXT YC
170 FOR XC=0 TO 14 STEP 2
172 X=INT(RND(1)*231)+24
174 POKE V+XC,X
176 NEXT XC
182 POKE V+21,255
185 FOR CR=0 TO 7
190 C=INT(RND(1)*16)+1
195 POKE V+(39+CR),C
200 NEXT CR
210 GOTO 185
999 REM SUBROUTINE FOLLOWS *********
1000 X=INT(RND(1)*255)+1
1010 Y=INT(RND(1)*255)+1
1015 FOR D=1 TO 50:NEXT D
1020 RETURN
9999 REM **NOTCHED WING BUTTERFLY
10000 DATA 3,0,192,57,129
10010 DATA 156,124,195,62,254
10020 DATA 102,127,255,36,255
10030 DATA 255,153,255,127,219
10040 DATA 254, 63,219,252, 31
10050 DATA 219,248, 15,219,240
10060 DATA  15,219,240, 31,219
10070 DATA 248, 63,219,252,127
10080 DATA 219,254,255,219,255
10090 DATA 255,219,255,255,219
10100 DATA 255,127,219,254,63
10110 DATA 153,252,31,24,248
10120 DATA 14,24,112
```

Line Number	Description
15	Initializes the address of the video chip, for future reference
30	Clears the screen, and disables any stray Sprites left on the screen
40	Sets the background and border screen colors
90-99	Sets the Sprite pointers for all eight Sprites to memory area 192

150-165	Steps through all the Y-coordinate registers and pokes each with a random Y value
170-176	Does the same for the X coordinate registers
182	Enables all eight Sprites
185-200	Generates a random color code for each of the eight Sprites and pokes these values into the color registers
210	This process is then repeated. The Sprites will continue to change colors until the program is stopped.

Horizontal/Vertical Expansion

Each Sprite can be expanded along the horizontal axis, along the vertical axis, or along both axes at once. This program shows all the possible combinations. If an expansion register contains zero, then neither Sprite is expanded. If it contains a 1, then Sprite #0 is expanded. A 2 means that Sprite #1 is expanded. A 3 (1+2) means both are expanded.

```
10 REM **HORIZONTAL/VERTICAL EXPANSION
15 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281,0 :POKE 53280, 2
50 V=53248
90 POKE 2040,192
92 POKE 2041,193
100 FOR S = 0 TO 1
110 FOR M=0 TO 62
120 READ A
130 POKE (192+S)*64+M,A
140 NEXT M
150 NEXT S
160 POKE V+39,6
170 POKE V+40,7
180 POKE V+0,100:POKE V+1,100
190 POKE V+2,160:POKE V+3,160
200 POKE V+21,3
205 FOR D=1 TO 500:NEXT D
210 FOR VE = 0 TO 3
220 FOR HE = 0 TO 3
230 POKE V+23,VE
```

```
235 FOR D=1 TO 500:NEXT D
240 POKE V+29,HE
245 FOR D=1 TO 500:NEXT D
250 NEXT HE
260 NEXT VE
9999 REM **MEAN WOLF SPRITE
10000 DATA 0,0,0,0,0,0,0
10010 DATA 0,48,1,192,112
10020 DATA 0,240,240,0,121
10030 DATA 224,0,61,192,0
10040 DATA 31,192,0,15,224
10050 DATA 0,63,224,255,255
10060 DATA 240,255,193,248,255
10070 DATA 255,252,170,255,254
10080 DATA 0,255,254,0,255
10090 DATA 254,0,255,254,170
10100 DATA 255,252,255,255,248
10110 DATA 1,255,240,1,255,240
10120 REM BLOCK SPRITE
10130 DATA 255,255,255
10140 DATA 255,255,255,255,255
10150 DATA 247,239,255,231,231
10160 DATA 255,199,227,255,135
10170 DATA 225,255,7,224,254
10180 DATA 7,224,124,7,255
10200 DATA 199,255,255,199,255
10210 DATA 255,199,255,224,124
10220 DATA 7,224,254,7,225
10230 DATA 255,7,227,255,135
10240 DATA 231,255,199,239,255
10250 DATA 231,255,255,247,255
10260 DATA 255,255,255,255,255
```

Line Number	**Description**
15	Initializes the address of the video chip, for future reference
30	Clears the screen, and disables any stray Sprites left on the screen
40	Sets the background and border screen colors

50	It won't cause a problem, but this line was not needed
90	Sets the Sprite pointer for Sprite #0 to memory area 192
92	Sets the Sprite pointer for Sprite #1 to memory area 193
100-150	This nested do-loop reads the values for the two Sprites and pokes these values into the proper memory locations
160-170	Sets the color codes
180-190	Assigns the X and Y coordinates for the two Sprites
205	A delay loop which leaves the image on the screen long enough for you to see it
210-260	This nested do-loop steps the vertical and horizontal expansion registers through all the possible values from 0 to 3

Multi-Color Sprites

Multi-color Sprites can use up to four colors per figure, counting the background color which shows through the transparent portion of the shape. In this example, two of the Sprites use four colors, and one uses only three.

Once you have run the program, try poking the value zero into register V+28. The resultant mess is what your multi-color Sprites will look like if you forget to define them as being multi-color.

```
10 REM **MULTI-COLOR SPRITES
15 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281,13:POKE 53280,5
90 POKE 2040,192
95 POKE 2041,193
100 POKE 2042,194
105 FOR S = 0 TO 2
110 FOR M=0 TO 62
120 READ A
130 POKE (192+S)*64+M,A
140 NEXT M
```

```
145 NEXT S
160 POKE V+39,6
170 POKE V+40,8
180 POKE V+41,2
190 POKE V+37,1
200 POKE V+38,0
210 POKE V+28,7
220 POKE V+0,150:POKE V+1,75
230 POKE V+2,150:POKE V+3,150
240 POKE V+4,150:POKE V+5,200
245 POKE V+23,7:POKE V+29,7
250 POKE V+21,7
9999 REM **MULTICOLOR BUTTERFLY
10000 DATA 8,0,128,10,2,128
10010 DATA 2,138,0,64,136
10020 DATA 5,80,168,21,84
10030 DATA 32,93,117,33,125
10040 DATA 125,101,253,127,103
10050 DATA 253,123,103,173,123
10060 DATA 103,173,123,103,173
10070 DATA 123,103,173,123,103
10080 DATA 173,123,103,173,123
10090 DATA 103,173,123,103,173
10100 DATA 127,103,253,95,101
10110 DATA 253,21,97,117,5
10120 DATA 32,84
10130 REM ** MULTICOLOR SAILBOAT
10140 DATA 0,8,0
10150 DATA 0,56,0,0,248
10160 DATA 0,3,248,0,15
10170 DATA 248,0,63,248,0
10180 DATA 255,248,0,0,8
10190 DATA 0,0,8,0,0
10200 DATA 8,0,0,8,0
10210 DATA 85,89,84,85,85
10220 DATA 85,255,255,255,255
10230 DATA 255,252,255,255,240
10240 DATA 252,0,0,252,0
10250 DATA 0,240,0,0,192
10260 DATA 0,0,0,0,0
10270 REM ** MULTICOLOR TARGET
10280 DATA 170,170
```

```
10290 DATA 170,170,170,170,149
10300 DATA 85,86,149,85,86
10310 DATA 159,255,246,159,255
10320 DATA 246,158,170,182,158
10330 DATA 170,182,158,150,182
10340 DATA 158,150,182,158,150
10350 DATA 182,158,150,182,158
10360 DATA 150,182,158,170,182
10370 DATA 158,170,182,159,255
10380 DATA 246,159,255,246,149
10390 DATA 85,86,149,85,86
10400 DATA 170,170,170,170,170,170
```

Line Number	Description
15	Initializes the address of the video chip, for future reference
30	Clears the screen, and disables any stray Sprites left on the screen
40	Sets the background and border screen colors
90-100	Sets the Sprite pointers for the three memory areas which will be used
105-145	Reads the data values for Sprites #0, 1, and 2 and pokes them into memory
160-180	Sets the Sprite colors for the three multi-color Sprites
190	Sets multi-color 1 which will be used by all three Sprites
200	Sets multi-color 2 which will be used by all three Sprites
210	Defines the three Sprites as being multi-colored. This is the sum of the place values of the Sprites (1+2+4)
220-240	Sets the X and Y coordinates for each Sprite
245	Expands the Sprites in both directions
250	Enables the three Sprites, so they will appear on the screen

Inverse Multi-Color Sprites

This useful program shows what happens when you poke the inverse of each of your data values into memory. This means that instead of poking the data value itself, you poke in 255 minus the data value.

On the Sprite grid, you can visualize the inverse of a Sprite as having a colored dot where every blank dot would be, and a blank dot where every colored dot would be. In binary numbers, this would mean a 1 in place of every zero, and a zero in place of every 1. To arrive at the decimal value of this number, just subtract the original value from 255. (If you are still not convinced, try an example yourself.)

In this program, the normal Sprites appear on the left, and their "inverse" forms appear on the right. Changing the colors will change both forms at the same time, so you can compare the results.

```
10 REM ** INVERSE MULTI-COLOR SPRITES
15 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281,13:POKE 53280,5
90 POKE 2040,192
95 POKE 2041,193
96 POKE 2042,194
97 POKE 2043,195
98 POKE 2044,196
100 POKE 2045,197
105 FOR S = 0 TO 2
110 FOR M=0 TO 62
120 READ A
130 POKE (192+S)*64+M,A
132 POKE (195+S)*64+M,(255-A)
140 NEXT M
145 NEXT S
160 POKE V+39,6
170 POKE V+40,8
180 POKE V+41,2
190 POKE V+37,1
200 POKE V+38,0
210 POKE V+28,63
```

```
220 POKE V+0,150:POKE V+1,75
230 POKE V+2,150:POKE V+3,150
240 POKE V+4,150:POKE V+5,200
241 POKE V+6,250:POKE V+7,75
242 POKE V+8,250:POKE V+9,150
243 POKE V+10,250:POKE V+11,200
245 POKE V+23,63:POKE V+29,63
250 POKE V+21,63
260 PRINT CHR$(147)
300 INPUT "BUTTERFLY SPRITE COLOR";BT
305 IF BT > 16 THEN GOTO 300
310 POKE V+39,BT:POKE V+42,BT
315 PRINT CHR$(147)
320 INPUT "SAILBOAT SPRITE COLOR";BT
330 IF BT > 16 THEN GOTO 320
340 POKE V+40,BT:POKE V+43,BT
345 PRINT CHR$(147)
350 INPUT "TARGET SPRITE COLOR";BT
360 IF BT > 16 THEN GOTO 350
370 POKE V+41,BT:POKE V+44,BT
375 PRINT CHR$(147)
380 INPUT "MULTI-COLOR 1";BT
390 IF BT > 16 THEN GOTO 380
400 POKE V+37,BT
405 PRINT CHR$(147)
410 INPUT "MULTI-COLOR 2";BT
420 IF BT > 16 THEN GOTO 410
430 POKE V+38,BT
435 PRINT CHR$(147)
440 INPUT "BACKGROUND COLOR";BT
445 IF BT = 14 THEN POKE 53281,3:GOTO 465
450 IF BT > 16 THEN GOTO 440
460 POKE 53281,BT
465 PRINT CHR$(147)
470 INPUT "BORDER COLOR";BT
480 IF BT > 16 THEN GOTO 470
490 POKE 53280,BT
9999 REM **MULTICOLOR BUTTERFLY
10000 DATA 8,0,128,10,2,128
10010 DATA 2,138,0,64,136
10020 DATA 5,80,168,21,84
10030 DATA 32,93,117,33,125
```

```
10040 DATA 125,101,253,127,103
10050 DATA 253,123,103,173,123
10060 DATA 103,173,123,103,173
10070 DATA 123,103,173,123,103
10080 DATA 173,123,103,173,123
10090 DATA 103,173,123,103,173
10100 DATA 127,103,253,95,101
10110 DATA 253,21,97,117,5
10120 DATA 32,84
10130 REM ** MULTICOLOR SAILBOAT
10140 DATA 0,8,0
10150 DATA 0,56,0,0,248
10160 DATA 0,3,248,0,15
10170 DATA 248,0,63,248,0
10180 DATA 255,248,0,0,8
10190 DATA 0,0,8,0,0
10200 DATA 8,0,0,8,0
10210 DATA 85,89,84,85,85
10220 DATA 85,255,255,255,255
10230 DATA 255,252,255,255,240
10240 DATA 252,0,0,252,0
10250 DATA 0,240,0,0,192
10260 DATA 0,0,0,0,0
10270 REM ** MULTICOLOR TARGET
10280 DATA 170,170
10290 DATA 170,170,170,170,149
10300 DATA 85,86,149,85,86
10310 DATA 159,255,246,159,255
10320 DATA 246,158,170,182,158
10330 DATA 170,182,158,150,182
10340 DATA 158,150,182,158,150
10350 DATA 182,158,150,182,158
10360 DATA 150,182,158,170,182
10370 DATA 158,170,182,159,255
10380 DATA 246,159,255,246,149
10390 DATA 85,86,149,85,86
10400 DATA 170,170,170,170,170,170
```

Line Number	**Description**
15	Initializes the address of the video chip, for future reference

30	Clears the screen, and disables any stray Sprites left on the screen
40	Sets the background and border screen colors
90-100	Sets the Sprite pointers for the six Sprites we will be using
105-145	Notice the sly trick we've pulled here. Although we have only enough data values for three different Sprites, we are filling TWO memory areas with each set of data values. For one memory area, we poke in the original value. In the other, we poke in (255-A) to achieve the "inverse" Sprite.
160-200	Sets the colors
210	Defines all six Sprites as multi-colored
220-243	Sets the screen coordinates for all six Sprites
245	Expands all six Sprites in both directions
250	Enables all six Sprites
260-490	Asks for your input in choosing the colors for Sprites, multi-color 1, multi-color 2, screen background and borders.
445	Note that I did not permit you to choose a background color identical to the one in which the questions are printed, or you couldn't read the last question.

Sprite to Sprite Priorities

Overlapping Sprites on the screen gives a feeling of depth to your graphics programs. In this example, note how the numbering of the Sprites was crucial, since they had to overlap in a certain pattern of dominance.

```
10 REM *** SPRITE TO SPRITE PRIORITIES
20 V=53248
25 PRINT CHR$(147):POKE V+21,0
40 POKE 2040,192
42 POKE 2041,193
44 POKE 2042,193
46 POKE 2043,194
48 POKE 2044,195
50 POKE 2045,193
```

```
52 POKE 2046,196
60 POKE 2047,197
62 FOR P = 0 TO 5
64 FOR Y=0 TO 62:READ A:POKE(192+P)*64+Y,
   A:NEXT Y
66 NEXT P
70 POKE 53280,0 :POKE 53281,0
80 POKE V+39,2
90 POKE V+40,8
100 POKE V+41,7
110 POKE V+42,6
120 POKE V+43,6
130 POKE V+44,7
140 POKE V+45,8
150 POKE V+46,2
155 X1=24:X2=135:Y=50 :A=30
160 POKE V+0,X1 :POKE V+1,Y
170 POKE V+2,X1+A :POKE V+3,Y+A
180 POKE V+4,X1+(2*A):POKE V+5,Y+(2*A)
190 POKE V+6,X1+(3*A):POKE V+7,Y+(3*A)
200 POKE V+8,X2       :POKE V+9,Y
210 POKE V+10,X2+(1*A) :POKE V+11,Y+A
220 POKE V+12,X2+(2*A):POKE V+13,Y+(2*A)
230 POKE V+14,X2+(3*A):POKE V+15,Y+(3*A)
240 POKE V+23,255:POKE V+29,255
250 POKE V+21,255
9999 REM LETTER G
10000 DATA 255
10010 DATA 255,255,255,255,255
10020 DATA 224,0,7,224,0
10030 DATA 7,231,255,231,231
10040 DATA 255,231,231,255,255
10050 DATA 231,255,255,231,255
10060 DATA 255,231,255,255,231
10070 DATA 255,255,231,240,7
10080 DATA 231,240,7,231,255
10090 DATA 231,231,255,231,231
10100 DATA 255,231,231,255,231
10110 DATA 224,0,7,224,0
10120 DATA 7,255,255,255,255
10130 DATA 255,255
```

```
10135 REM LETTER O
10140 DATA 255,255,255
10150 DATA 255,255,255,224,0
10160 DATA 7,224,0,7,224
10170 DATA 0,7,227,255,199
10180 DATA 227,255,199,227,255
10190 DATA 199,227,255,199,227
10200 DATA 255,199,227,255,199
10210 DATA 227,255,199,227,255
10220 DATA 199,227,255,199,227
10230 DATA 255,199,227,255,199
10240 DATA 224,0,7,224,0
10250 DATA 7,224,0,7,255
10260 DATA 255,255,255,255,255
10270 REM LETTER D
10280 DATA 255,255,255,255,255
10290 DATA 255,224,0,7,224
10300 DATA 0,7,249,255,231
10310 DATA 249,255,231,249,255
10320 DATA 231,249,255,231,249
10330 DATA 255,231,249,255,231
10340 DATA 249,255,231,249,255
10350 DATA 231,249,255,231,249
10360 DATA 255,231,249,255,231
10370 DATA 249,255,231,249,255
10380 DATA 231,224,0,7,224
10390 DATA 0,7,255,255,255
10400 DATA 255,255,255
10405 REM LETTER J
10410 DATA 255,255
10420 DATA 255,255,255,255,224
10430 DATA 0,7,224,0,7
10440 DATA 255,227,255,255,227
10450 DATA 255,255,227,255,255
10460 DATA 227,255,255,227,255
10470 DATA 255,227,255,255,227
10480 DATA 255,255,227,255,231
10490 DATA 227,255,231,227,255
10500 DATA 231,227,255,231,227
10510 DATA 255,231,227,255,224
10520 DATA 3,255,224,3,255
10530 DATA 255,255,255,255,255
```

```
10540 DATA 255
10545 REM LETTER B
10550 DATA 255,255,255,255
10560 DATA 255,255,192,0,3
10570 DATA 192,0,3,243,255
10580 DATA 243,243,255,243,243
10590 DATA 255,243,243,255,243
10600 DATA 243,255,243,240,0
10610 DATA 3,240,0,31,240
10620 DATA 0,3,243,255,243
10630 DATA 243,255,243,243,255
10640 DATA 243,243,255,243,243
10650 DATA 255,243,192,0,3
10660 DATA 192,0,3,255,255
10670 DATA 255,255,255,255
10675 REM EXCLAMATION POINT
10680 DATA 255
10690 DATA 255,255,255,255,255
10700 DATA 255,231,255,255,231
10710 DATA 255,255,231,255,255
10720 DATA 231,255,255,231,255
10730 DATA 255,231,255,255,231
10740 DATA 255,255,231,255,255
10750 DATA 231,255,255,231,255
10760 DATA 255,231,255,255,231
10770 DATA 255,255,231,255,255
10780 DATA 231,255,255,255,255
10790 DATA 255,231,255,255,231
10800 DATA 255,255,255,255,255
10810 DATA 255,255
```

Line Number	Description
20	Initializes the address of the video chip, for future reference
25	Clears the screen, and disables any stray Sprites left on the screen
40-60	Sets each of the Sprite pointers to its own memory area (notice that Sprites #1, 2, and 5 will be the same)

62-66	Reads the data values and pokes them into memory
70	Sets the screen background and border colors
80-150	Sets the color for each Sprite
155-230	The X and Y coordinates for this program were set up so the spaces between the Sprites on the screen could be changed with one variable. A sketch of this screen layout and how it was planned can be found elsewhere in this book, if you are having trouble visualizing the results
240	All eight Sprites are expanded in both directions
250	All eight Sprites are enabled.

Sprite Priorities in Motion

Watching Sprites move in relation to one another gives an even better feeling for how Sprite to Sprite priorities work. As these Sprites pass through the same screen area, observe the dominance hierarchy in action as they pass in front of or behind each other.

```
10 REM ** MOVING SPRITE TO SPRITE PRIORITIES
15 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281,0 :POKE 53280, 2
50 V=53248
60 FOR SN=0 TO 5
80 POKE 2040+SN,192
90 NEXT SN
110 FOR M=0 TO 62
120 READ A
130 POKE 192*64+M,A
140 NEXT M
150 X=225:Y=220
155 POKE V+23,63:POKE V+29,63
160 POKE V+39,2
170 POKE V+40,7
171 POKE V+41,8
172 POKE V+42,4
173 POKE V+43,5
174 POKE V+44,6
```

```
180 POKE V+0,50  :POKE V+1,Y
190 POKE V+2,75  :POKE V+3,Y+20
191 POKE V+4,150:POKE V+5,Y+30
192 POKE V+6,X   :POKE V+7,50
193 POKE V+8,X+15:POKE V+9,100
194 POKE V+10,X+30:POKE V+11,80
200 POKE V+21,63
205 X=X-2:Y=Y-3
210 IF X<31 OR Y<31 THEN GOTO 230
220 GOTO 180
230 END
9999 REM **MEAN WOLF SPRITE
10000 DATA 0,0,0,0,0,0,0
10010 DATA 0,48,1,192,112
10020 DATA 0,240,240,0,121
10030 DATA 224,0,61,192,0
10040 DATA 31,192,0,15,224
10050 DATA 0,63,224,255,255
10060 DATA 240,255,193,248,255
10070 DATA 255,252,170,255,254
10080 DATA 0,255,254,0,255
10090 DATA 254,0,255,254,170
10100 DATA 255,252,255,255,248
10110 DATA 1,255,240,1,255,240
```

Line Number	Description
15	Initializes the address of the video chip, for future reference
30	Clears the screen, and disables any stray Sprites left on the screen
40	Sets the background and border screen colors
50	This line is superfluous. (That's a polite way of saying I goofed.)
60-90	Sets the Sprite pointer for all six Sprites to the same memory area
110-140	Reads the data values and pokes them into memory
150	Initializes the values for the X and Y coordinates
155	Expands all six Sprites along both axes
160-174	Sets the Sprite colors

180-220 Plots the Sprites on the screen, and increments the X and Y values to produce movement

Sprite to Background Priorities

Changing the values in the Sprite Background Priority register will enable Sprites to pass behind background figures. This program shows a simple example.

Notice how slowly the eight Sprites move when all must be moved, one after another. This points out that for complicated action games involving Sprites, you will need to use machine language routines to achieve truly fast and furious action.

```
10 REM ** SPRITE TO BACKGROUND PRIORITIES
15 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281, 3:POKE 53280,6
90 POKE 2040,192
92 POKE 2041,192
94 POKE 2042,192
96 POKE 2043,192
97 POKE 2044,192
98 POKE 2045,192
99 POKE 2046,192
100 POKE 2047,192
110 FOR M=0 TO 62
120 READ A
130 POKE 192*64+M,A
140 NEXT M
142 FOR Z=1 TO 15:PRINT:NEXT Z
144 FOR Z=1 TO 200
146 PRINT "#";
148 NEXT Z
149 POKE V+27,31
150 FOR YC=1 TO 15 STEP 2
160 Y=INT(RND(1)*175)+50
162 POKE V+YC,Y
165 NEXT YC
170 FOR XC=0 TO 14 STEP 2
172 X=INT(RND(1)*231)+24
174 POKE V+XC,X
176 NEXT XC
```

```
182 POKE V+21,255
185 FOR CR=0 TO 7
190 C=INT(RND(1)*16)+1
195 POKE V+(39+CR),C
200 NEXT CR
210 FOR YC=1 TO 15 STEP 2
220 POKE V+YC,PEEK(V+YC) +1
225 IF PEEK(V+YC) > 254 THEN POKE V+YC,1
230 NEXT YC
240 GOTO 210
900 END
999 REM SUBROUTINE FOLLOWS *********
1000 X=INT(RND(1)*255)+1
1010 Y=INT(RND(1)*255)+1
1015 FOR D=1 TO 50:NEXT D
1020 RETURN
9999 REM **NOTCHED WING BUTTERFLY
10000 DATA 3,0,192,57,129
10010 DATA 156,124,195,62,254
10020 DATA 102,127,255,36,255
10030 DATA 255,153,255,127,219
10040 DATA 254, 63,219,252, 31
10050 DATA 219,248, 15,219,240
10060 DATA  15,219,240, 31,219
10070 DATA 248, 63,219,252,127
10080 DATA 219,254,255,219,255
10090 DATA 255,219,255,255,219
10100 DATA 255,127,219,254,63
10110 DATA 153,252,31,24,248
10120 DATA 14,24,112
```

Line Number	Description
15	Initializes the address of the video chip, for future reference
30	Clears the screen, and disables any stray Sprites left on the screen
40	Sets the background and border screen colors
90-100	Sets the Sprite pointer for each of the eight Sprites to memory area 192

110-140	Reads the data values and pokes them into memory
142-148	Prints a background figure on the screen
149	Turns "on" the bits in the Sprite Background Priority register for Sprites #0 through 4
150-165	Sets the X coordinates for each Sprite to random values
170-176	Sets the Y coordinates for each Sprite to random values
182	Enables all eight Sprites
185-200	Sets random color codes for each of the eight Sprites
210-230	Moves each Sprite downward in the Y direction by looking at the existing Y value and adding 1 to it

Sprite to Sprite Collisions

Sprite collisions with other Sprites are detected by looking at the contents of a register. This program does the looking for you, then prints a message to show when a collision is taking place.

```
10 REM ** SPRITE TO SPRITE COLLISIONS
15 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281, 1:POKE 53280, 2
50 V=53248
90 POKE 2040,192
100 POKE 2041,192
110 FOR M=0 TO 62
120 READ A
130 POKE 192*64+M,A
140 NEXT M
160 POKE V+39,0
162 POKE V+40,9
163 POKE V+23,3:POKE V+29,3
165 FOR A = 50 TO 255
168 IF PEEK(V+30) AND 1 = 1 THEN GOTO 170
170 POKE V+0,A:POKE V+1,A
180 POKE V+2,(255-A):POKE V+3,A
185 POKE V+21,3
```

```
190 IF PEEK(V+30) AND 1 = 1 THEN GOSUB 500
290 NEXT A
300 END
500 POKE V+39,4:PRINT "COLLISION! ";
505 POKE V+40,7
510 RETURN
9999 REM ** HAPPY RABBIT SPRITE
10000 DATA 0,0,0
10010 DATA 0,0,0,0,0
10020 DATA 0,31,199,240,27
10030 DATA 199,176,24,238,48
10040 DATA 24,238,48,24,238
10050 DATA 48,24,255,48,25
10060 DATA 255,176,3,147,192
10070 DATA 7,147,224,15,147
10080 DATA 240,31,239,248,31
10090 DATA 255,248,15,255,240
10100 DATA 7,199,224,3,199
10110 DATA 192,1,199,128,0
10120 DATA 127,0,0,0,0
```

Line Number	Description
15	Initializes the address of the video chip, for future reference
30	Clears the screen, and disables any stray Sprites left on the screen
40	Sets the background and border screen colors
50	Oops! Goofed again.
90-100	Sets the Sprite pointers for Sprites #0 and #1
110-140	Reads the data values and pokes them into memory
160-162	Sets the colors
163	Expands both Sprites along both axes
165-290	This loop moves the Sprites toward each other. Each time they move closer, the Sprite Collision Register is examined. If the PEEK shows that the bit for Sprite #0 (place value 1) has been turned on, then the program branches to the subroutine

168 This is a dummy statement to PEEK into register V+30 and thus reset the value of that register to zero. (It may not have been zero when the program started.)

500-510 This subroutine changes the color of Sprite #0 and prints the "collision" message on the screen

Sprite to Background Collisions

Detecting collisions between Sprites and background figures is done in much the same way we detect collisions between Sprites. This program keeps you posted on the contents of the register involved, so you can see how it changes as the two Sprites come in contact with a vertical "wall" on the screen.

Notice how the values in the register reflect that just Sprite #0 is in contact with the wall (V+31=1), or just Sprite #1 is in contact (V+31=2), or they are both in contact with the wall (V+31=1+2=3).

```
10 REM ** SPRITE TO BACKGROUND COLLISIONS
15 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281, 1:POKE 53280, 2
41 FOR T = 1 TO 23
42 PRINT "                 ♦♦♦♦♦"
43 NEXT T
45 PRINT "GREEN RABBIT IS SPRITE #0"
46 PRINT "RED RABBIT IS SPRITE #1"
50 V=53248
90 POKE 2040,192
100 POKE 2041,192
110 FOR M=0 TO 62
120 READ A
130 POKE 192*64+M,A
140 NEXT M
160 POKE V+39,5
162 POKE V+40,2
163 POKE V+23,3:POKE V+29,3
165 FOR A = 255 TO 1 STEP -1
168 PRINT "ョ"
169 PRINT "V+31= "; PEEK(V+31)
```

```
170 POKE V+0,A:POKE V+1,130
180 POKE V+2,(255-A):POKE V+3,78
185 POKE V+21,3
190 REM IF PEEK(V+31) AND 2 = 2 THEN
    GOSUB 500
290 NEXT A
300 GOTO 165
500 POKE V+40,0
510 RETURN
9999 REM ** HAPPY RABBIT SPRITE
10000 DATA 0,0,0
10010 DATA 0,0,0,0,0
10020 DATA 0,31,199,240,27
10030 DATA 199,176,24,238,48
10040 DATA 24,238,48,24,238
10050 DATA 48,24,255,48,25
10060 DATA 255,176,3,147,192
10070 DATA 7,147,224,15,147
10080 DATA 240,31,239,248,31
10090 DATA 255,248,15,255,240
10100 DATA 7,199,224,3,199
10110 DATA 192,1,199,128,0
10120 DATA 127,0,0,0,0
```

Line Number	Description
15	Initializes the address of the video chip, for future reference
30	Clears the screen, and disables any stray Sprites left on the screen
40	Sets the background and border screen colors
41-46	Prints the "wall" and a message on the screen
50	Oops again!
90-100	Sets the Sprite pointers for the two Sprites we will be using. Both point to the same memory area.
110-140	Reads the data values and pokes them into the memory locations
160-162	Sets the color codes
163	Expands both Sprites in both directions

165-290 Moves the two Sprites toward the "wall" from opposite sides of the screen. The contents of register V+31 are printed on the screen, and if the bit for Sprite #1 (place value 2) is turned on, the program branches to the subroutine.

500-510 This subroutine changes the color of Sprite #1.

All Sorts of Collisions

In this program, you can watch both kinds of collisions happen. The contents of register V+30 will be printed, so you can observe the changes as the Sprites collide. The contents of register V+31 will also be shown, so you can watch the Sprites hit the background figure. To make things a bit more interesting, one of the Sprites has been defined as having a lower priority than the background, and will pass behind the background figure.

```
10 REM ** SPRITE TO BACKGROUND COLLISIONS
15 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281, 1:POKE 53280, 2
41 FOR T = 1 TO 23
42 PRINT "                    ●●●●●"
43 NEXT T
45 PRINT "GREEN RABBIT IS SPRITE #0"
46 PRINT "RED RABBIT IS SPRITE #1"
50 V=53248
90 POKE 2040,192
100 POKE 2041,192
110 FOR M=0 TO 62
120 READ A
130 POKE 192*64+M,A
140 NEXT M
160 POKE V+39,5
162 POKE V+40,2
163 POKE V+23,3:POKE V+29,3
164 POKE V+27,1
165 FOR A = 255 TO 1 STEP -1
168 PRINT "ヨ":PRINT "V+30= ";PEEK(V+30)
169 PRINT "V+31= "; PEEK(V+31)
```

```
170 POKE V+0,A:POKE V+1,100
180 POKE V+2,(255-A):POKE V+3,78
185 POKE V+21,3
190 REM IF PEEK(V+31) AND 2 = 2 THEN
    GOSUB 500
290 NEXT A
300 GOTO 165
500 POKE V+40,0
510 RETURN
9999 REM ** HAPPY RABBIT SPRITE
10000 DATA 0,0,0
10010 DATA 0,0,0,0,0
10020 DATA 0,31,199,240,27
10030 DATA 199,176,24,238,48
10040 DATA 24,238,48,24,238
10050 DATA 48,24,255,48,25
10060 DATA 255,176,3,147,192
10070 DATA 7,147,224,15,147
10080 DATA 240,31,239,248,31
10090 DATA 255,248,15,255,240
10100 DATA 7,199,224,3,199
10110 DATA 192,1,199,128,0
10120 DATA 127,0,0,0,0
```

Line Number	Description
15	Initializes the address of the video chip, for future reference
30	Clears the screen, and disables any stray Sprites left on the screen
40-46	Prints the background figure and messages on the screen
50	This must be my favorite line. I seem to use it everywhere.
90-100	Sets the Sprite pointers
110-140	Reads the data values and pokes them into memory
160-162	Sets the colors
163	Expands both Sprites along both axes

164	Sets the bit for Sprite #0 in the Background Priority register, so Sprite #0 will pass behind any background figures
165-290	Prints the values of both collision registers as the Sprites move across the screen
500-510	This subroutine changes the color of Sprite #1 if it has been involved in a Sprite-to-background collision

Animation by Switching Sprite Pointers

This animation process defines the Sprite on the screen, then switches the Sprite pointer to different memory areas containing different versions of the Sprite shape.

```
10 REM ANIMATION BY SWITCHING POINTERS
20 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281,13:POKE 53280,6
50 FOR SN = 0 TO 7
60 POKE 2040+SN,192
70 FOR M=0 TO 62
80 READ A
90 POKE (192+SN)*64+M,A
100 NEXT M
110 NEXT SN
120 FOR YC=1 TO 15 STEP 2
130 POKE V+YC,200
140 NEXT YC
145 X=24
150 FOR XC=0 TO 14 STEP 2
160 POKE V+XC,X
170 X=X+30
180 NEXT XC
182 POKE V+39,2
183 POKE V+40,8
184 POKE V+41,7
185 POKE V+42,5
186 POKE V+43,6
187 POKE V+44,4
```

```
188 POKE V+45,0
189 POKE V+46,1
190 POKE V+23,255:POKE V+29,255
200 POKE V+21,255
210 FOR SP=7 TO 0 STEP -1
215 FOR SN=0 TO 7
220 POKE 2040+SP,192+SN
225 FOR D=1 TO 200:NEXT D
230 NEXT SN
240 NEXT SP
9999 REM FLOWER #1
10000 DATA 0,0,0,0,0,0
10010 DATA 0,0,0,0,0
10020 DATA 0,0,0,0,0
10030 DATA 0,0,0,0,0
10040 DATA 0,0,0,0,0
10050 DATA 0,0,0,0,0
10060 DATA 0,0,0,0,0
10070 DATA 0,0,0,0,0
10080 DATA 0,0,124,0,0
10090 DATA 124,0,0,124,0
10100 DATA 0,56,0,0,16
10110 DATA 0,0,16,0,0
10120 DATA 16,0
10130 REM FLOWER #2
10140 DATA 0,0,0
10150 DATA 0,0,0,0,0
10160 DATA 0,0,0,0,0
10170 DATA 0,0,0,0,0
10180 DATA 0,0,0,0,0
10190 DATA 0,0,0,0,0
10200 DATA 0,0,0,0,0
10210 DATA 0,124,0,0,124
10220 DATA 0,0,124,0,0
10230 DATA 56,0,0,16,0
10240 DATA 0,16,0,0,16
10250 DATA 0,0,16,0,0
10260 DATA 16,0,0,16,0
10270 REM FLOWER #3
10280 DATA 0,0,0,0,0
10290 DATA 0,0,0,0,0
10300 DATA 0,0,0,0,0
```

```
10310 DATA 0,0,0,0,0
10320 DATA 0,0,124,0,0
10330 DATA 124,0,0,124,0
10340 DATA 0,124,0,0,56
10350 DATA 0,0,16,0,0
10360 DATA 16,0,0,16,0
10370 DATA 0,16,0,0,16
10380 DATA 0,0,16,0,0
10390 DATA 16,0,0,16,0
10400 DATA 0,16,0
10405 REM FLOWER #3
10410 DATA 0,0
10420 DATA 0,0,0,0,0
10430 DATA 0,0,0,0,0
10440 DATA 0,124,0,0,124
10450 DATA 0,0,124,0,0
10460 DATA 124,0,0,56,0
10470 DATA 7,17,192,3,147
10480 DATA 128,3,215,128,1
10490 DATA 255,0,0,254,0
10500 DATA 0,124,0,0,16
10510 DATA 0,0,16,0,0
10520 DATA 16,0,0,16,0
10530 DATA 0,16,0,0,16
10540 DATA 0
10545 REM FLOWER #4
10550 DATA 0,0,0,0
10560 DATA 0,0,0,124,0
10570 DATA 0,254,0,1,255
10580 DATA 0,1,255,0,1
10590 DATA 255,0,1,255,0
10600 DATA 0,56,0,7,17
10610 DATA 192,3,147,128,3
10620 DATA 215,128,1,255,0
10630 DATA 0,254,0,0,124
10640 DATA 0,0,16,0,0
10650 DATA 16,0,0,16,0
10660 DATA 0,16,0,0,16
10670 DATA 0,0,16,0
10680 REM FLOWER #5
10690 DATA 0,0,0,0,0,0
10700 DATA 4,84,64,2,84
```

```
10710 DATA 128,1,85,0,1
10720 DATA 255,0,1,255,0
10730 DATA 1,255,0,0,56
10740 DATA 0,7,17,192,3
10750 DATA 147,128,3,215,128
10760 DATA 1,255,0,0,254
10770 DATA 0,0,124,0,0
10780 DATA 16,0,0,16,0
10790 DATA 0,16,0,0,16
10800 DATA 0,0,16,0,0
10810 DATA 16,0
10815 REM FLOWER #6
10820 DATA 23,57,208
10830 DATA 11,187,160,53,255
10840 DATA 88,26,254,176,13
10850 DATA 125,96,7,255,192
10860 DATA 3,255,128,1,255
10870 DATA 0,0,56,0,7
10880 DATA 17,192,3,147,128
10890 DATA 3,215,128,1,255
10900 DATA 0,0,254,0,0
10910 DATA 124,0,0,16,0
10920 DATA 0,16,0,0,16
10930 DATA 0,0,16,0,0
10940 DATA 16,0,0,16,0
10950 REM FLOWER #7
10960 DATA 255,255,255,202,170
10970 DATA 166,117,125,92,58
10980 DATA 186,184,29,85,112
10990 DATA 15,255,224,7,255
11000 DATA 192,3,255,128,0
11010 DATA 56,0,15,17,240
11020 DATA 15,147,224,7,215
11030 DATA 192,3,255,128,1
11040 DATA 255,0,0,124,0
11050 DATA 0,16,0,0,16
11060 DATA 0,0,16,0,0
11070 DATA 16,0,0,16,0
11080 DATA 0,16,0
11085 REM FLOWER #8
11090 DATA 255,255
11100 DATA 255,128,0,1,128
```

```
11110 DATA 0,1,128,0,1
11120 DATA 130,0,33,135,0
11130 DATA 113,141,128,217,152
11140 DATA 193,141,176,65,5
11150 DATA 128,0,1,135,195
11160 DATA 225,132,66,33,135
11170 DATA 219,225,128,24,1
11180 DATA 128,0,1,131,255
11190 DATA 193,132,0,33,136
11200 DATA 0,17,144,0,9
11210 DATA 128,0,1,255,255
11220 DATA 255
```

Line Number	Description
20	Initializes the address of the video chip, for future reference
30	Clears the screen, and disables any stray Sprites left on the screen
40	Sets the background and border screen colors
50-110	This nested loop points each Sprite to memory area 192, and reads the data values into the eight memory areas
120-140	This loop sets the Y coordinates for all eight Sprites to 200
145-180	The X coordinates for the eight Sprites are spaced at intervals 30 dots apart, beginning with value 24
182-189	Sets the Sprite colors
190	Expands all eight Sprites along both axes
200	Enables all the Sprites
210-240	This set of nested do-loops is the heart of the program. Each Sprite in turn is animated by rotating its Sprite Pointer through the eight versions of the Sprite shape stored in the different memory areas numbered 192 through 199.
225	This delay loop is necessary so that each different image will stay on the screen long enough for your eyes to see it.

More Animation by Switching Pointers

This example animates just one Sprite, but the motion is run both forwards and "backwards."

```
10 REM ANIMATION BY SWITCHING POINTERS
20 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281,1:POKE 53280,2
60 FOR SN=0 TO 6
70 FOR M = 0 TO 62
80 READ A
90 POKE (192+SN)*64+M,A
100 NEXT M
110 NEXT SN
120 POKE V+23,1:POKE V+29,1
125 POKE V+39,0
130 POKE V+0,135:POKE V+1,150
140 POKE V+21,1
150 FOR SN = 192 TO 198
160 POKE 2040,SN
165 FOR D = 1 TO 500 :NEXT D
170 NEXT SN
180 FOR SN = 198 TO 192 STEP -1
190 POKE 2040,SN
200 FOR D = 1 TO 500 :NEXT D
210 NEXT SN
10000 REM FACE #1
11230 DATA 255,255,255,128
11240 DATA 0,1,128,0,1
11250 DATA 128,0,1,130,0
11260 DATA 33,135,0,113,141
11270 DATA 128,217,152,193,141
11280 DATA 128,0,1,128,0
11290 DATA 1,135,195,225,132
11300 DATA 66,33,135,219,225
11310 DATA 128,24,1,128,0
11320 DATA 1,135,255,225,140
11330 DATA 0,49,152,0,25
11340 DATA 128,0,1,128,0
11350 DATA 1,255,255,255
```

```
11355 REM FACE #2
11360 DATA 255
11370 DATA 255,255,128,0,1
11380 DATA 128,0,1,128,0
11390 DATA 1,128,0,1,135
11400 DATA 0,113,141,128,217
11410 DATA 152,193,141,128,0
11420 DATA 1,135,195,225,135
11430 DATA 195,225,132,66,33
11440 DATA 135,219,225,128,24
11450 DATA 1,128,0,1,135
11460 DATA 255,225,140,0,49
11470 DATA 128,0,1,128,0
11480 DATA 1,128,0,1,255
11490 DATA 255,255
11495 REM FACE #3
11500 DATA 255,255,255
11510 DATA 128,0,1,128,0
11520 DATA 1,128,0,1,128
11530 DATA 0,1,135,0,113
11540 DATA 141,128,217,152,193
11550 DATA 141,128,0,1,135
11560 DATA 195,225,132,66,33
11570 DATA 132,66,33,135,219
11580 DATA 225,128,24,1,128
11590 DATA 0,1,135,255,225
11600 DATA 128,0,1,128,0
11610 DATA 1,128,0,1,128
11620 DATA 0,1,255,255,255
11630 REM FACE #4
11640 DATA 255,255,255,128,0
11650 DATA 1,128,0,1,128
11660 DATA 0,1,128,0,1
11670 DATA 128,0,1,128,0
11680 DATA 1,159,193,253,128
11690 DATA 0,1,135,195,225
11700 DATA 132,66,33,132,66
11710 DATA 33,135,219,225,128
11720 DATA 24,1,128,0,1
11730 DATA 128,0,1,128,0
11740 DATA 1,135,255,225,128
11750 DATA 0,1,128,0,1
```

```
11760 DATA 255,255,255
11765 REM FACE #5
11770 DATA 255,255
11780 DATA 255,128,0,1,128
11790 DATA 0,1,128,0,1
11800 DATA 128,0,1,128,0
11810 DATA 1,159,128,253,159
11820 DATA 193,253,128,0,1
11830 DATA 135,195,225,132,66
11840 DATA 33,132,66,33,135
11850 DATA 219,225,128,24,1
11860 DATA 128,0,1,152,0
11870 DATA 25,140,0,49,132
11880 DATA 0,33,135,255,225
11890 DATA 128,0,1,255,255
11900 DATA 255
11905 REM FACE #6
11910 DATA 255,255,255,128
11920 DATA 0,1,128,0,1
11930 DATA 128,0,1,131,195
11940 DATA 225,132,0,17,136
11950 DATA 0,9,144,0,5
11960 DATA 128,0,1,135,195
11970 DATA 225,135,195,225,132
11980 DATA 66,33,135,219,225
11990 DATA 128,24,1,128,0
12000 DATA 1,152,0,25,140
12010 DATA 0,49,132,0,33
12020 DATA 135,255,225,128,0
12030 DATA 1,255,255,255
12035 REM FACE #7
12040 DATA 255
12050 DATA 255,255,128,0,1
12060 DATA 128,0,1,128,0
12070 DATA 1,128,0,1,135
12080 DATA 195,241,140,0,25
12090 DATA 152,0,13,128,0
12100 DATA 1,135,195,225,132
12110 DATA 66,33,132,66,33
12120 DATA 135,219,225,128,24
12130 DATA 1,184,0,29,156
12140 DATA 0,57,142,0,113
```

```
12150 DATA 135,255,225,131,255
12160 DATA 193,128,0,1,255
12170 DATA 255,255
```

Line Number	Description
20	Initializes the address of the video chip, for future reference
30	Clears the screen, and disables any stray Sprites left on the screen
40	Sets the background and border screen colors
60-110	Reads the data values and pokes them into the memory locations
120	Expands Sprite #0 in both directions
125	Sets the color for Sprite #0
130	Sets the X and Y values for Sprite #0
140	Enables Sprite #0
150-170	Steps the pointer values from memory locations 192 through 199
180-210	Steps the pointer values backwards through memory locations 199 to 192

Animation by Enabling Sprites in Turn

Sprites can also be animated by defining all eight Sprites to the same screen location, then enabling them one at a time. If each Sprite is a slightly different shape, you will see "motion" as the result.

```
10 REM ANIMATION BY ENABLING SPRITES IN TURN
20 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281,1 :POKE 53280,5
50 FOR SN = 0 TO 7
60 POKE 2040+SN,192+SN
70 FOR M=0 TO 62
80 READ A
90 POKE (192+SN)*64+M,A
100 NEXT M
110 NEXT SN
```

```
120 FOR YC=1 TO 15 STEP 2
130 POKE V+YC,200
140 NEXT YC
145 X=24
150 FOR XC=0 TO 14 STEP 2
160 POKE V+XC,150
180 NEXT XC
182 FOR CR = 0 TO 7
183 POKE V+(39+CR),2
184 NEXT CR
185 POKE V+23,255:POKE V+29,255
186 POKE V+21,1
187 FOR SP=0 TO 7
188 POKE V+21,2↑SP
189 FOR D=1 TO 200:NEXT D
190 NEXT SP
9999 REM FLOWER #1
10000 DATA 0,0,0,0,0,0
10010 DATA 0,0,0,0,0
10020 DATA 0,0,0,0,0
10030 DATA 0,0,0,0,0
10040 DATA 0,0,0,0,0
10050 DATA 0,0,0,0,0
10060 DATA 0,0,0,0,0
10070 DATA 0,0,0,0,0
10080 DATA 0,0,124,0,0
10090 DATA 124,0,0,124,0
10100 DATA 0,56,0,0,16
10110 DATA 0,0,16,0,0
10120 DATA 16,0
10130 REM FLOWER #2
10140 DATA 0,0,0
10150 DATA 0,0,0,0,0
10160 DATA 0,0,0,0,0
10170 DATA 0,0,0,0,0
10180 DATA 0,0,0,0,0
10190 DATA 0,0,0,0,0
10200 DATA 0,0,0,0,0
10210 DATA 0,124,0,0,124
10220 DATA 0,0,124,0,0
10230 DATA 56,0,0,16,0
10240 DATA 0,16,0,0,16
```

```
10250 DATA 0,0,16,0,0
10260 DATA 16,0,0,16,0
10270 REM FLOWER #3
10280 DATA 0,0,0,0,0
10290 DATA 0,0,0,0,0
10300 DATA 0,0,0,0,0
10310 DATA 0,0,0,0,0
10320 DATA 0,0,124,0,0
10330 DATA 124,0,0,124,0
10340 DATA 0,124,0,0,56
10350 DATA 0,0,16,0,0
10360 DATA 16,0,0,16,0
10370 DATA 0,16,0,0,16
10380 DATA 0,0,16,0,0
10390 DATA 16,0,0,16,0
10400 DATA 0,16,0
10405 REM FLOWER #3
10410 DATA 0,0
10420 DATA 0,0,0,0,0
10430 DATA 0,0,0,0,0
10440 DATA 0,124,0,0,124
10450 DATA 0,0,124,0,0
10460 DATA 124,0,0,56,0
10470 DATA 7,17,192,3,147
10480 DATA 128,3,215,128,1
10490 DATA 255,0,0,254,0
10500 DATA 0,124,0,0,16
10510 DATA 0,0,16,0,0
10520 DATA 16,0,0,16,0
10530 DATA 0,16,0,0,16
10540 DATA 0
10545 REM FLOWER #4
10550 DATA 0,0,0,0
10560 DATA 0,0,0,124,0
10570 DATA 0,254,0,1,255
10580 DATA 0,1,255,0,1
10590 DATA 255,0,1,255,0
10600 DATA 0,56,0,7,17
10610 DATA 192,3,147,128,3
10620 DATA 215,128,1,255,0
10630 DATA 0,254,0,0,124
10640 DATA 0,0,16,0,0
```

```
10650 DATA 16,0,0,16,0
10660 DATA 0,16,0,0,16
10670 DATA 0,0,16,0
10680 REM FLOWER #5
10690 DATA 0,0,0,0,0,0
10700 DATA 4,84,64,2,84
10710 DATA 128,1,85,0,1
10720 DATA 255,0,1,255,0
10730 DATA 1,255,0,0,56
10740 DATA 0,7,17,192,3
10750 DATA 147,128,3,215,128
10760 DATA 1,255,0,0,254
10770 DATA 0,0,124,0,0
10780 DATA 16,0,0,16,0
10790 DATA 0,16,0,0,16
10800 DATA 0,0,16,0,0
10810 DATA 16,0
10815 REM FLOWER #6
10820 DATA 23,57,208
10830 DATA 11,187,160,53,255
10840 DATA 88,26,254,176,13
10850 DATA 125,96,7,255,192
10860 DATA 3,255,128,1,255
10870 DATA 0,0,56,0,7
10880 DATA 17,192,3,147,128
10890 DATA 3,215,128,1,255
10900 DATA 0,0,254,0,0
10910 DATA 124,0,0,16,0
10920 DATA 0,16,0,0,16
10930 DATA 0,0,16,0,0
10940 DATA 16,0,0,16,0
10950 REM FLOWER #7
10960 DATA 255,255,255,202,170
10970 DATA 166,117,125,92,58
10980 DATA 186,184,29,85,112
10990 DATA 15,255,224,7,255
11000 DATA 192,3,255,128,0
11010 DATA 56,0,15,17,240
11020 DATA 15,147,224,7,215
11030 DATA 192,3,255,128,1
11040 DATA 255,0,0,124,0
11050 DATA 0,16,0,0,16
```

```
11060 DATA 0,0,16,0,0
11070 DATA 16,0,0,16,0
11080 DATA 0,16,0
11085 REM FLOWER #8
11090 DATA 255,255
11100 DATA 255,128,0,1,128
11110 DATA 0,1,128,0,1
11120 DATA 130,0,33,135,0
11130 DATA 113,141,128,217,152
11140 DATA 193,141,176,65,5
11150 DATA 128,0,1,135,195
11160 DATA 225,132,66,33,135
11170 DATA 219,225,128,24,1
11180 DATA 128,0,1,131,255
11190 DATA 193,132,0,33,136
11200 DATA 0,17,144,0,9
11210 DATA 128,0,1,255,255
11220 DATA 255
```

Line Number	Description
20	Initializes the address of the video chip
30	Clears the screen and disables any stray Sprites on the screen
40	Sets the screen and border colors
50-110	Sets the Sprite pointer, reads in the data values, and pokes them into memory for each of the eight Sprites
120-140	Steps through the Y coordinate registers for each Sprite and assigns them all to Y value 200
145	This line is left over from another program (oops!)
150-180	Assigns the same X coordinate value to each of the eight Sprites
182-184	Assigns the same color code to all the Sprites
185	Expands all the Sprites in both directions
186	Enables Sprite #0
187-190	This loop enables each of the Sprites in turn (only one Sprite is enabled at any one time), then delays long enough for you to see the newly enabled Sprite on the screen

Producing Inverse Single Color Sprites

I call a Sprite "inverse" if the picture portion of the Sprite is made up of the "holes" instead of the colored dots. If you have data values for a normal Sprite, it is easy to create an inverse Sprite without having to redo the data values. You simply poke 255 minus the data value into the memory locations. (How this trick works is explained in the program about producing inverse multi-color Sprites, found earlier in this chapter.)

This is a rather interesting example, since the Sprites used here were actually designed to look inverse when the data values were determined. Therefore when you use this "inverse" process, they become "inverse inverse" or normal! (You can see the original version of these Sprites in the program on Sprite to Sprite priorities found in this chapter.)

```
10 REM *** PRODUCING INVERSE SPRITES
20 V=53248
25 PRINT CHR$(147):POKE V+21,0
40 POKE 2040,192
42 POKE 2041,193
44 POKE 2042,193
46 POKE 2043,194
48 POKE 2044,195
50 POKE 2045,193
52 POKE 2046,196
60 POKE 2047,197
62 FOR P = 0 TO 5
64 FOR Y=0 TO 62:READ A:POKE(192+P)*64+Y,
   (255-A):NEXT Y
66 NEXT P
70 POKE 53280,0 :POKE 53281,0
80 POKE V+39,2
90 POKE V+40,8
100 POKE V+41,7
110 POKE V+42,6
120 POKE V+43,6
130 POKE V+44,7
140 POKE V+45,8
150 POKE V+46,2
155 X1=24:X2=135:Y=50 :A=30
```

```
160 POKE V+0,X1  :POKE V+1,Y
170 POKE V+2,X1+A  :POKE V+3,Y+A
180 POKE V+4,X1+(2*A):POKE V+5,Y+(2*A)
190 POKE V+6,X1+(3*A):POKE V+7,Y+(3*A)
200 POKE V+8,X2      :POKE V+9,Y
210 POKE V+10,X2+(1*A)  :POKE V+11,Y+A
220 POKE V+12,X2+(2*A):POKE V+13,Y+(2*A)
230 POKE V+14,X2+(3*A):POKE V+15,Y+(3*A)
240 POKE V+23,255:POKE V+29,255
250 POKE V+21,255
9999 REM LETTER G
10000 DATA 255
10010 DATA 255,255,255,255,255
10020 DATA 224,0,7,224,0
10030 DATA 7,231,255,231,231
10040 DATA 255,231,231,255,255
10050 DATA 231,255,255,231,255
10060 DATA 255,231,255,255,231
10070 DATA 255,255,231,240,7
10080 DATA 231,240,7,231,255
10090 DATA 231,231,255,231,231
10100 DATA 255,231,231,255,231
10110 DATA 224,0,7,224,0
10120 DATA 7,255,255,255,255
10130 DATA 255,255
10135 REM LETTER O
10140 DATA 255,255,255
10150 DATA 255,255,255,224,0
10160 DATA 7,224,0,7,224
10170 DATA 0,7,227,255,199
10180 DATA 227,255,199,227,255
10190 DATA 199,227,255,199,227
10200 DATA 255,199,227,255,199
10210 DATA 227,255,199,227,255
10220 DATA 199,227,255,199,227
10230 DATA 255,199,227,255,199
10240 DATA 224,0,7,224,0
10250 DATA 7,224,0,7,255
10260 DATA 255,255,255,255,255
10270 REM LETTER D
10280 DATA 255,255,255,255,255
10290 DATA 255,224,0,7,224
```

```
10300 DATA 0,7,249,255,231
10310 DATA 249,255,231,249,255
10320 DATA 231,249,255,231,249
10330 DATA 255,231,249,255,231
10340 DATA 249,255,231,249,255
10350 DATA 231,249,255,231,249
10360 DATA 255,231,249,255,231
10370 DATA 249,255,231,249,255
10380 DATA 231,224,0,7,224
10390 DATA 0,7,255,255,255
10400 DATA 255,255,255
10405 REM LETTER J
10410 DATA 255,255
10420 DATA 255,255,255,255,224
10430 DATA 0,7,224,0,7
10440 DATA 255,227,255,255,227
10450 DATA 255,255,227,255,255
10460 DATA 227,255,255,227,255
10470 DATA 255,227,255,255,227
10480 DATA 255,255,227,255,231
10490 DATA 227,255,231,227,255
10500 DATA 231,227,255,231,227
10510 DATA 255,231,227,255,224
10520 DATA 3,255,224,3,255
10530 DATA 255,255,255,255,255
10540 DATA 255
10545 REM LETTER B
10550 DATA 255,255,255,255
10560 DATA 255,255,192,0,3
10570 DATA 192,0,3,243,255
10580 DATA 243,243,255,243,243
10590 DATA 255,243,243,255,243
10600 DATA 243,255,243,240,0
10610 DATA 3,240,0,31,240
10620 DATA 0,3,243,255,243
10630 DATA 243,255,243,243,255
10640 DATA 243,243,255,243,243
10650 DATA 255,243,192,0,3
10660 DATA 192,0,3,255,255
10670 DATA 255,255,255,255
10675 REM EXCLAMATION POINT
10680 DATA 255
```

```
10690 DATA 255,255,255,255,255
10700 DATA 255,231,255,255,231
10710 DATA 255,255,231,255,255
10720 DATA 231,255,255,231,255
10730 DATA 255,231,255,255,231
10740 DATA 255,255,231,255,255
10750 DATA 231,255,255,231,255
10760 DATA 255,231,255,255,231
10770 DATA 255,255,231,255,255
10780 DATA 231,255,255,255,255
10790 DATA 255,231,255,255,231
10800 DATA 255,255,255,255,255
10810 DATA 255,255
```

Line Number	Description
20	Initializes the address of the video chip, for future reference
25	Clears the screen, and disables any stray Sprites left on the screen
40-60	Sets the Sprite pointers for all eight Sprites. Notice that three of the eight point to the same area.
62-66	Reads the data values and pokes them into memory. Note that the value being poked is (255-A) instead of A. This gives the inverse image results.
70	Sets the screen background and border colors
80-150	Sets the Sprite colors
155-230	This group of statements sets the X and Y coordinates for the Sprites in terms of a variable amount of space between them on the screen. This allows you to change that spacing easily by just changing the value of A.
240	Expands all Sprites in both directions
250	Enables all Sprites

Producing "Double" Sprites

Sometime you may have a Sprite shape in mind that is just too large to squeeze into a 24 by 21 dot grid. Then you might consider defining two Sprites that you will move around the

screen as a unit. However, as you run this program, you will see that it is impossible to move both Sprites at the exact same moment. This leaves you with a sort of "see-saw" effect, where one Sprite always moves first, then the other catches up.

```
10 REM ** DOUBLE SPRITES
15 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281,3:POKE 53280,6
90 POKE 2040,192
92 POKE 2041,193
110 FOR SN=0 TO 1
115 FOR M=0 TO 62
120 READ A
130 POKE (192+SN)*64+M,A
135 NEXT M
140 NEXT SN
160 POKE V+39,6
170 POKE V+40,6
180 POKE V+0,100
190 POKE V+2,148
195 POKE V+23,3:POKE V+29,3
210 FOR Y = 0 TO 255
220 POKE V+1,Y:POKE V+3,Y
230 POKE V+21,3
240 NEXT Y
9999 REM HOUSE--LEFT SIDE
10000 DATA 0
10010 DATA 31,255,0,60,0
10020 DATA 0,126,0,0,255
10030 DATA 0,1,255,128,3
10040 DATA 255,192,7,255,224
10050 DATA 15,255,240,31,255
10060 DATA 248,63,255,252,127
10070 DATA 255,254,255,255,255
10080 DATA 128,0,1,128,0
10090 DATA 1,128,0,1,129
10100 DATA 255,1,129,255,1
10110 DATA 129,255,1,129,253
10120 DATA 1,129,255,1,255
10130 DATA 255,255
```

```
10135 REM HOUSE--RIGHT SIDE
10140 DATA 255,248,0
10150 DATA 0,12,0,0,6
10160 DATA 0,0,3,0,0
10170 DATA 1,128,0,0,192
10180 DATA 0,0,96,0,0
10190 DATA 48,0,0,24,0
10200 DATA 0,12,0,0,6
10210 DATA 255,255,255,0,0
10220 DATA 1,0,0,1,0
10230 DATA 0,1,62,124,249
10240 DATA 34,68,137,34,68
10250 DATA 137,62,124,249,0
10260 DATA 0,1,255,255,255
```

Line Number	Description
15	Initializes the address of the video chip
30	Clears the screen and disables any stray Sprites on the screen
40	Sets the screen and border colors
90-92	Sets the Sprite pointers for Sprites #0 and #1
110-140	Reads the data values and pokes them into memory
160-170	Sets the Sprite colors
180-190	The heart of this program is positioning the two Sprites correctly in relation to each other. Since these Sprites are both expanded, they are 48 dots wide. Therefore, the difference in their X coordinates must always be 48 if you want to move them around the screen and have them remain side by side.
195	Expand both Sprites in both directions
210-240	Increments the Y coordinates to move both Sprites down the screen

Reach Out and Touch a Sprite

This little program uses the collision register to figure out when the boy and girl Sprites have joined hands.

```
10 REM ** POSITIONING SPRITES
15 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281,1 :POKE 53280, 3
90 POKE 2040,192
100 POKE 2041,193
105 FOR SN = 0 TO 1
110 FOR M=0 TO 62
120 READ A
130 POKE (192+SN)*64+M,A
140 NEXT M
145 NEXT SN
160 POKE V+39,9
165 POKE V+40,9
168 POKE V+23,3:POKE V+29,3
170 POKE V+1,150
175 POKE V+3,150
178 IF PEEK(V+30) AND 3 = 3 THEN GOTO 180
180 FOR X = 1 TO 200
185 POKE V+0,X
190 POKE V+2,(200-X)
191 POKE V+21,3
192 IF PEEK(V+30) AND 3 = 3 THEN END
195 NEXT X
210 END
9999 REM **MAN IN SWIM SUIT
10000 DATA 0,62,0,0,42
10010 DATA 0,0,62,0,0
10020 DATA 28,0,0,127,0
10030 DATA 1,255,192,3,62
10040 DATA 96,6,62,48,28
10050 DATA 54,56,8,62,16
10060 DATA 0,0,0,0,62
10070 DATA 0,0,62,0,0
10080 DATA 54,0,0,0,0
10090 DATA 0,54,0,0,54
10100 DATA 0,0,54,0,0
10110 DATA 54,0,0,54,0
10120 DATA 1,247,128
10130 REM WOMAN IN SWIM SUIT
10140 DATA 0,62
10150 DATA 0,0,170,128,0
```

```
10160 DATA 255,128,0,28,0
10170 DATA 0,127,0,1,193
10180 DATA 192,3,54,96,6
10190 DATA 62,48,28,62,56
10200 DATA 8,62,16,0,127
10210 DATA 0,0,255,128,0
10220 DATA 0,0,0,54,0
10230 DATA 0,54,0,0,54
10240 DATA 0,0,54,0,0
10250 DATA 54,0,0,54,0
10260 DATA 0,54,0,1,247
10270 DATA 128
```

Line Number	Description
15	Initializes the beginning address of the video chip
30	Clears the screen and disables any stray Sprites on the screen
40	Sets the screen background and border colors
90-100	Sets the Sprite pointers for Sprites #0 and 1
105-145	Reads in the data values and pokes them into memory
160-165	Sets the colors for both Sprites
168	Expands both Sprites in both directions
170-175	Sets the Y coordinate values for each of the two Sprites
178	This statement resets the values in the Sprite collision register, just in case the register was non-zero when the program began executing.
180-195	This loop changes the X value to move the Sprites closer together on the screen
192	Checks the contents of the Sprite collision register to see if Sprites #0 and #1 have overlapped (place values 1 and 2 would be "on")

Puzzling with Sprites

Combining a Sprite with its inverse image can make interesting visual effects. Here we have the original Sprite meeting its inverse Sprite.

```
10 REM ** "PUZZLE" SPRITES
15 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281,1 :POKE 53280, 3
90 POKE 2040,192
100 POKE 2041,193
110 FOR M=0 TO 62
120 READ A
130 POKE 192*64+M,255-A
135 POKE 193*64+M,A
140 NEXT M
160 POKE V+39,5
165 POKE V+40,6
168 POKE V+23,3:POKE V+29,3
170 POKE V+1,150
175 POKE V+3,150
178 IF PEEK(V+30) AND 3 = 3 THEN GOTO 180
180 FOR X = 1 TO 100
185 POKE V+0,X
190 POKE V+2,(200-X)
191 POKE V+21,3
195 NEXT X
210 END
9999 REM **MAN IN SWIM SUIT
10000 DATA 0,62,0,0,42
10010 DATA 0,0,62,0,0
10020 DATA 28,0,0,127,0
10030 DATA 1,255,192,3,62
10040 DATA 96,6,62,48,28
10050 DATA 54,56,8,62,16
10060 DATA 0,0,0,0,62
10070 DATA 0,0,62,0,0
10080 DATA 54,0,0,0,0
10090 DATA 0,54,0,0,54
10100 DATA 0,0,54,0,0
10110 DATA 54,0,0,54,0
10120 DATA 1,247,128
```

Line Number	Description
15	Initializes the beginning address of the video chip

30	Clears the screen and disables any stray Sprites on the screen
40	Sets the screen border and background colors
90-100	Sets the Sprite pointers
110-140	We are poking the values for the original Sprite and inverse Sprite at the same time, while reading only one set of data values.
160-165	Sets the Sprite colors
168	Expands both Sprites in both directions
170-175	Sets the Y coordinates for both Sprites
178	This line is not needed
180	I cheated and stopped the action while X was 100, instead of having the program detect the relative position of the two Sprites.

Puzzle Pieces from Two Sprites

This interesting example shows two Sprites designed to fit together as puzzle pieces do.

```
10 REM ** PUZZLE PIECES FROM TWO SPRITES
15 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281,13:POKE 53280,5
90 POKE 2040,193
100 POKE 2041,192
105 FOR SN=0 TO 1
110 FOR M=0 TO 62
120 READ A
130 POKE (192+SN)*64+M,A
140 NEXT M
145 NEXT SN
160 POKE V+39,7
165 POKE V+40,5
168 POKE V+23,3:POKE V+29,3
170 POKE V+1,150
175 POKE V+3,150
178 IF PEEK(V+30) AND 3 = 3 THEN GOTO 180
180 FOR X=1 TO 200
185 POKE V+0,X
190 POKE V+2,(200-X)
```

```
191 POKE V+21,3
192 IF PEEK(V+30) AND 3 = 3 THEN END
195 NEXT X
10270 REM PUZZLE--RIGHT SIDE
10280 DATA 255,255,255,255,255
10290 DATA 255,1,255,255,127
10300 DATA 255,255,127,255,255
10310 DATA 127,255,255,7,255
10320 DATA 255,63,255,255,63
10330 DATA 255,255,63,255,255
10340 DATA 63,255,255,0,127
10350 DATA 255,0,127,255,15
10360 DATA 255,255,15,255,255
10370 DATA 15,255,255,255,255
10380 DATA 255,255,255,255,255
10390 DATA 255,255,255,255,255
10400 DATA 255,255,255
10405 REM PUZZLE--LEFT SIDE
10410 DATA 255,254
10420 DATA 0,255,254,0,255
10430 DATA 255,252,255,255,0
10440 DATA 255,255,0,255,255
10450 DATA 0,255,255,240,255
10460 DATA 255,128,255,255,128
10470 DATA 255,255,128,255,255
10480 DATA 128,255,255,255,255
10490 DATA 255,255,255,255,224
10500 DATA 255,255,224,255,255
10510 DATA 224,255,254,0,255
10520 DATA 254,0,255,254,0
10530 DATA 255,254,0,255,254
10540 DATA 0
```

Line Number	Description
15	Initializes the beginning address of the video chip
30	Clears the screen and disables any stray Sprites on the screen
40	Sets the screen border and background colors
90-100	Sets the Sprite pointers

105-145	Reads in the data values and pokes them into memory
160-165	Sets the colors
168	Expands both Sprites in both directions
170-175	Sets the Y coordinates
178	This statement takes care of the collision register being non-zero when the program starts
180-195	This loop moves the "puzzle pieces" closer together and checks to see if they are touching.

The Sprite Sampler

This is a very handy program to have available when your neighbor asks you what Sprites can really do. My favorite part is when the wolf chases the rabbit in the animation sequence.

```
10 REM (C) 1983 BY SALLY GREENWOOD LARSEN
20 REM SPRITE GRAPHICS SAMPLER
31 PRINT CHR$(31)
32 V=53248
33 GOSUB 20000
34 POKE V+23,0:POKE V+29,0
40 GOSUB 20000
41 POKE 53280,2
42 PRINTTAB(200)"  THIS IS THE SPRITE GRAPHICS
   SAMPLER"
43 PRINT:PRINT:PRINT "           PLEASE STAND
   BY"
44 PRINT:PRINT "       WHILE I LOAD THE DATA
   VALUES"
46 POKE 2040,192
47 POKE 2041,193
48 POKE 2042,194
49 POKE 2043,195
50 POKE 2044,196
51 POKE 2045,197
52 POKE 2046,198
53 POKE 2047,199
58 FOR Y=0 TO 62:READ A:POKE 192*64+Y,A:NEXT Y
60 FOR Y=0 TO 62:READ A:POKE 193*64+Y,A:NEXT Y
70 FOR Y=0 TO 62:READ A:POKE 194*64+Y,A:NEXT Y
```

```
80 FOR Y=0 TO 62:READ A:POKE 195*64+Y,A:NEXT Y
82 FOR Y=0 TO 62:READ A:POKE 196*64+Y,A:NEXT Y
84 FOR Y=0 TO 62:READ A:POKE 197*64+Y,A:NEXT Y
86 FOR Y=0 TO 62:READ A:POKE 198*64+Y,A:NEXT Y
88 FOR Y=0 TO 62:READ A:POKE 199*64+Y,A:NEXT Y
910 GOSUB 20000
920 POKE 53280,4
930 PRINT:PRINT " USING SPRITE GRAPHICS, I
    CAN DEFINE"
935 PRINT " ONE SPRITE..."
940 POKE V+39,05
950 POKE V+0,100:POKE V+01,100
960 POKE V+21,1
980 GOSUB 15000
1000 GOSUB 20000
1010 POKE 53280,7
1020 PRINT CHR$(147):PRINT  " OR MANY SPRITES..."
1030 POKE V+39,5
1031 POKE V+40,5
1032 POKE V+41,5
1033 POKE V+42,5
1034 POKE V+43,5
1035 POKE V+44,5
1036 POKE V+45,5
1037 POKE V+46,5
1040 POKE V+00,50:POKE V+01,100
1050 POKE V+10,100:POKE V+11,100
1060 POKE V+04,150:POKE V+05,100
1070 POKE V+14,200:POKE V+15,100
1080 POKE V+08,50:POKE V+09,200
1090 POKE V+02,100:POKE V+03,200
1100 POKE V+12,150:POKE V+13,200
1110 POKE V+06,200:POKE V+07,200
1115 POKE V+23,85:POKE V+29,85
1120 POKE V+21,255
1125 GOSUB 15000
1130 PRINT CHR$(147):PRINT "I CAN CHANGE THE
     COLOR OF THE SPRITES..."
1135 FOR C = 1 TO 8
1140 POKE V+39,C
1145 GOSUB 25000
1150 POKE V+40,C+1
```

```
1155 GOSUB 25000
1160 POKE V+41,C+2
1165 GOSUB 25000
1170 POKE V+42,C+3
1175 GOSUB 25000
1180 POKE V+43,C+4
1185 GOSUB 25000
1190 POKE V+44,C+5
1195 GOSUB 25000
1200 POKE V+45,C+6
1205 GOSUB 25000
1210 POKE V+46,C+7
1215 GOSUB 25000
1218 NEXT C
1220 PRINT CHR$(147):PRINT:PRINT "THE
     BACKGROUND COLOR..."
1250 FOR BA = 0 TO 15
1260 POKE 53281,BA
1265 FOR D=1 TO 300:NEXT D
1270 NEXT BA
1272 POKE 53281,1
1275 PRINT CHR$(147):PRINT:PRINT "        OR
     THE BORDER COLOR."
1276 FOR BO = 0 TO 15
1278 POKE 53280,BO
1279 FOR D=1 TO 300:NEXT D
1280 NEXT BO
1290 GOSUB 15000
1400 GOSUB 20000
1420 PRINT CHR$(147):PRINT:PRINT "I CAN MOVE
     THE SPRITES"
1422 PRINT "  HORIZONTALLY ACROSS THE SCREEN..."
1425 POKE V+41,5
1428 POKE V+43,2
1452 FOR X=1 TO 255 STEP  2
1454 POKE V+4,X:POKE V+5,100
1456 POKE V+8,X:POKE V+9,200
1457 POKE V+21,20
1458 NEXT X
1459 GOSUB 15000
1460 GOSUB 20000
1480 PRINT:PRINT "   OR MOVE THEM VERTICALLY."
```

```
1490 FOR Y=1 TO 200 STEP 2
1500 POKE V+4,100:POKE V+5,Y
1510 POKE V+8,200:POKE V+9,Y
1515 POKE V+21,20
1520 NEXT Y
1530 GOSUB 15000
1600 GOSUB 20000
1601 POKE 53280,14:POKE V+39,8:POKE V+46,4
1602 PRINT "I CAN EXPAND THE SPRITES"
1603 PRINT "  ALONG THE Y-AXIS..."
1611 POKE V+0,100:POKE V+1,150
1612 POKE V+14,200:POKE V+15,150
1620 POKE V+29,00:POKE V+23,00
1622 POKE V+21,129
1625 GOSUB 15000
1626 POKE V+29,00:POKE V+23,129
1627 GOSUB 15000
1628 PRINT:PRINT "         ALONG THE X-AXIS..."
1630 POKE V+23,00:POKE V+29,00
1635 GOSUB 15000
1636 POKE V+23,00:POKE V+29,129
1637 GOSUB 15000
1638 PRINT:PRINT "                   OR ALONG
     BOTH AXES..."
1640 POKE V+23,00:POKE V+29,00
1645 GOSUB 15000
1648 POKE V+23,129:POKE V+29,129
1649 GOSUB 15000
1700 GOSUB 20000
1705 FOR M = 1 TO 18:PRINT:NEXT M
1710 PRINT " I CAN PASS SPRITES"
1715 PRINT"     IN FRONT OF EACH OTHER"
1718 GOSUB 15000
1720 X=225:Y=220
1730 POKE V+23,255:POKE V+29,255
1731 POKE V+39,2
1732 POKE V+40,7
1733 POKE V+41,8
1734 POKE V+42,4
1735 POKE V+43,5
1736 POKE V+44,6
1740 POKE V+0,50 :POKE V+1,Y
```

```
1742 POKE V+2,75  :POKE V+3,Y+20
1744 POKE V+4,150 :POKE V+5,Y+30
1746 POKE V+6,X  :POKE V+7,50
1748 POKE V+8,X+15:POKE V+9,100
1750 POKE V+10,X+30:POKE V+11,80
1755 POKE V+21,63
1760 X=X-2:Y=Y-3:IF X<31  OR Y<31  THEN GOTO
     1780
1770 GOTO 1740
1780 GOSUB 15000
1800 GOSUB 20000
1900 PRINT "I CAN AUTOMATICALLY DETECT"
1905 PRINT " WHEN TWO SPRITES COLLIDE..."
1910 POKE V+30,0
1925 POKE V+23,3:POKE V+29,3
1935 POKE V+39,2:POKE V+40,5
1940 X=1
1950 POKE V+0,X:POKE V+1,200
1960 POKE V+2,250-X:POKE V+3,200
1961 POKE V+21,3
1962 IF PEEK(V+30)AND3=3 THEN GOSUB 26000:GOTO
     1995
1990 X = X+1:GOTO 1950
1995 GOSUB 15000
2000 GOSUB 20000
2010 POKE V+21,0
2999 FOR D= 1 TO 20:PRINT:NEXT D
3000 PRINT "I CAN ALSO DETECT"
3001 PRINT "A COLLISION WITH"
3002 PRINT "THE BACKGROUND..."
3003 PRINT
3004 PRINT CHR$(18)
3005 FOR D = 1 TO 40:PRINT CHR$(113);:NEXT D
3010 PRINT CHR$(146)
3020 POKE V+39,6
3030 Y=1
3035 POKE V+31,0
3040 POKE V+0,210:POKE V+1,Y
3045 POKE V+21,1
3050 IF PEEK(V+31) AND 1=1 THEN GOSUB 27000:
     GOTO 3070
3060 Y=Y+1:GOTO 3040
```

```
3070 GOSUB 15000
5000 GOSUB 20000
5010 PRINT " AND I CAN ANIMATE A SPRITE"
5020 PRINT:PRINT "  BY RAPIDLY SWITCHING
     DIFFERENT"
5030 PRINT:PRINT "    VERSIONS OF THE SAME
     SPRITE."
5038 POKE V+23,5:POKE V+29,5
5040 POKE V+41,11
5045 POKE V+39,4
5050 POKE V+5,150
5052 POKE V+1,150
5055 FOR X = 200 TO 95 STEP -6
5056 POKE V+21,5
5058 POKE V+4,X
5059 POKE V+0,X-50
5060 POKE 2042,194
5065 POKE 2040,192
5070 FOR D = 1 TO 100:NEXT D
5075 POKE 2042,195
5078 POKE 2040,193
5079 FOR D = 1 TO 200:NEXT D
5080 NEXT X
5090 GOSUB 15000
6000 GOSUB 20000
6010 PRINT CHR$(147)
6020 PRINT:PRINT:PRINT " THIS IS BUT A SAMPLE OF"
6030 PRINT:PRINT "  WHAT CAN BE DONE WITH
     SPRITES."
6040 PRINT:PRINT "   LEARN ALL THESE SKILLS"
6050 PRINT:PRINT "    AND MUCH MORE...BY READING"
6060 PRINT
6070 PRINT CHR$(156):PRINT "  SPRITE GRAPHICS
     FOR THE COMMODORE 64"
6080 PRINT "      BY SALLY GREENWOOD LARSEN"
6090 POKE V+23,0:POKE V+29,0
6500 POKE 2042,194
6510 POKE 2040,192
7000 POKE V+0,24 :POKE V+1,225
7010 POKE V+2,61 :POKE V+3,175
7020 POKE V+4,98 :POKE V+5,225
7030 POKE V+6,135:POKE V+7,175
```

```
7040 POKE V+8,172:POKE V+9,225
7050 POKE V+10,209:POKE V+11,175
7060 POKE V+12,246:POKE V+13,225
7065 POKE V+16,128
7070 POKE V+14,28 :POKE V+15,175
7071 POKE V+39,2
7072 POKE V+40,8
7073 POKE V+41,7
7074 POKE V+42,5
7075 POKE V+43,6
7076 POKE V+44,4
7078 POKE V+45,12
7079 POKE V+46,0
8000 POKE V+21,255
8050 GOSUB 15000
8060 GOSUB 15000
8065 GOSUB 15000
8070 PRINT CHR$(147)
9000 END
9999 REM SPRITE DATA FOR HAPPY RABBIT
10000 DATA 0
10010 DATA 0,0,255,0,255
10020 DATA 255,129,255,15,199
10030 DATA 224,3,199,128,0
10040 DATA 238,0,0,238,0
10050 DATA 0,238,0,0,255
10060 DATA 0,1,255,128,3
10070 DATA 147,192,7,147,224
10080 DATA 15,147,240,31,239
10090 DATA 248,31,125,248,15
10100 DATA 131,240,7,131,224
10110 DATA 3,131,192,1,255
10120 DATA 128,0,127,0,0
10130 DATA 0,0
10135 REM SPRITE DATA FOR SURPRISED RABBIT
10140 DATA 0,0,0
10150 DATA 0,0,0,0,0
10160 DATA 0,31,199,240,27
10170 DATA 199,176,24,238,48
10180 DATA 24,238,48,24,238
10190 DATA 48,24,255,48,25
10200 DATA 255,176,3,147,192
```

```
10210 DATA 7,147,224,15,147
10220 DATA 240,31,239,248,31
10230 DATA 255,248,15,255,240
10240 DATA 7,199,224,3,199
10250 DATA 192,1,199,128,0
10260 DATA 127,0,0,0,0
10270 REM SPRITE DATA FOR WOLF
10280 DATA 0,0,0,0,0
10290 DATA 0,0,0,48,1
10300 DATA 192,112,0,240,240
10310 DATA 0,121,224,0,61
10320 DATA 192,0,31,192,0
10330 DATA 15,224,0,63,224
10340 DATA 0,255,240,127,199
10350 DATA 248,255,255,252,255
10360 DATA 255,254,255,255,254
10370 DATA 128,255,254,42,255
10380 DATA 254,127,255,252,1
10390 DATA 255,248,1,255,240
10400 DATA 1,255,240
10405 REM SPRITE DATA FOR MEAN WOLF
10410 DATA 0,0
10420 DATA 0,0,0,0,0
10430 DATA 0,48,1,192,112
10440 DATA 0,240,240,0,121
10450 DATA 224,0,61,192,0
10460 DATA 31,192,0,15,224
10470 DATA 0,63,224,255,255
10480 DATA 240,255,193,248,255
10490 DATA 255,252,170,255,254
10500 DATA 0,255,254,0,255
10510 DATA 254,0,255,254,170
10520 DATA 255,252,255,255,248
10530 DATA 1,255,240,1,255
10540 DATA 240
```

```
14500 REM SUBROUTINES BEGIN HERE
15000 FOR D=1 TO 2000:NEXT D
15010 RETURN
20000 PRINT CHR$(147)
20005 POKE 53281,1
20008 POKE 53280,6
20010 POKE V+21,0
25000 FOR D = 1 TO 20:NEXT D
25010 RETURN
26000 PRINTTAB(160) "          K A B O O M ! ! !"
26010 FOR Q = 1 TO 5
26020 POKE 53280,0
26025 FOR D=1 TO 100:NEXT D
26030 POKE 53280,7
26040 FOR D=1 TO 100:NEXT D
26050 NEXT Q
26060 RETURN
27000 FOR Z = 1 TO 5
27020 POKE 53281,5
27030 FOR D = 1 TO 100:NEXT D
27040 POKE 53281,07
27050 FOR D=1 TO 100:NEXT D
27060 NEXT Z
27070 RETURN
27075 REM SUBROUTINES END HERE
29990 REM SPRITE DATA FOR LETTER A
30000 DATA 31
30010 DATA 255,240,31,255,240
30020 DATA 31,255,240,31,255
30030 DATA 240,30,0,240,30
30040 DATA 0,240,30,0,240
30050 DATA 30,0,240,31,255
30060 DATA 240,31,255,240,31
30070 DATA 255,240,30,0,240
30080 DATA 30,0,240,30,0
```

```
30090 DATA 240,30,0,240,30
30100 DATA 0,240,30,0,240
30110 DATA 30,0,240,30,0
30120 DATA 240,30,0,240,30
30130 DATA 0,240
30135 REM SPRITE DATA FOR LETTER B
30140 DATA 255,255,192
30150 DATA 255,255,224,255,255
30160 DATA 240,224,0,56,224
30170 DATA 0,24,224,0,24
30180 DATA 224,0,24,224,0
30190 DATA 48,224,0,96,255
30200 DATA 255,192,255,255,128
30210 DATA 255,255,192,224,0
30220 DATA 224,224,0,112,224
30230 DATA 0,48,224,0,48
30240 DATA 224,0,48,224,0
30250 DATA 112,255,255,224,255
30260 DATA 255,192,255,255,128
30270 REM SPRITE DATA FOR LETTER C
30280 DATA 255,255,255,255,255
30290 DATA 255,255,255,255,240
30300 DATA 0,3,240,0,3
30310 DATA 240,0,0,240,0
30320 DATA 0,240,0,0,240
30330 DATA 0,0,240,0,0
30340 DATA 240,0,0,240,0
30350 DATA 0,240,0,0,240
30360 DATA 0,0,240,0,0
30370 DATA 240,0,3,240,0
30380 DATA 3,240,0,3,255
30390 DATA 255,255,255,255,255
30400 DATA 255,255,255
30815 REM SPRITE DATA FOR BLOCK SPRITE
30820 DATA 255,255,255
```

```
30830 DATA 255,255,255,255,255
30840 DATA 247,239,255,231,231
30850 DATA 255,199,227,255,135
30860 DATA 225,255,7,224,254
30870 DATA 7,224,124,7,255
30880 DATA 199,255,255,199,255
30890 DATA 255,199,255,224,124
30900 DATA 7,224,254,7,225
30910 DATA 255,7,227,255,135
30920 DATA 231,255,199,239,255
30930 DATA 231,255,255,247,255
30940 DATA 255,255,255,255,255
```

CHAPTER 13

BINARY NOTATION AND BOOLEAN OPERATIONS

To reassure those of you who have never seen a binary number before, the binary number system follows all the same rules and patterns our familiar decimal numbering system does. You will find that you catch on quickly to how it works. The only difference is that while the decimal numbering system is based on TEN, the binary numbering system is based on TWO. Otherwise, they operate just alike.

Now, since computers are supposed to be so powerful and such excellent number-crunchers, you may wonder why we even bother to go to the trouble of using a special numbering system with them. Why not just work with decimal numbers, the ones we are used to using?

From a computer's point of view, the decimal system is a nightmare for data-handling purposes. It is very difficult for a machine which codes all its information in terms of electricity being "on" or "off" to find a way to uniquely represent all ten digits of the decimal system in a reliable and efficient fashion. What the computer needs is a numbering system that has only two digits — one of which can be represented by the electrical current in a given memory location being "off" and the other represented by the current being "on."

Before we go into a discussion of how binary numbers can be actually coded in the computer's memory, we need to examine the binary numbering system in more detail.

FAMILIAR GROUND — THE DECIMAL SYSTEM

First we'll put into words what we intuitively understand about how the decimal system works.

The decimal system (also called Base 10) is based on the number 10. It has ten different digits — 0 through 9. Each of the place values in a decimal number is worth ten times the value of the place before it:

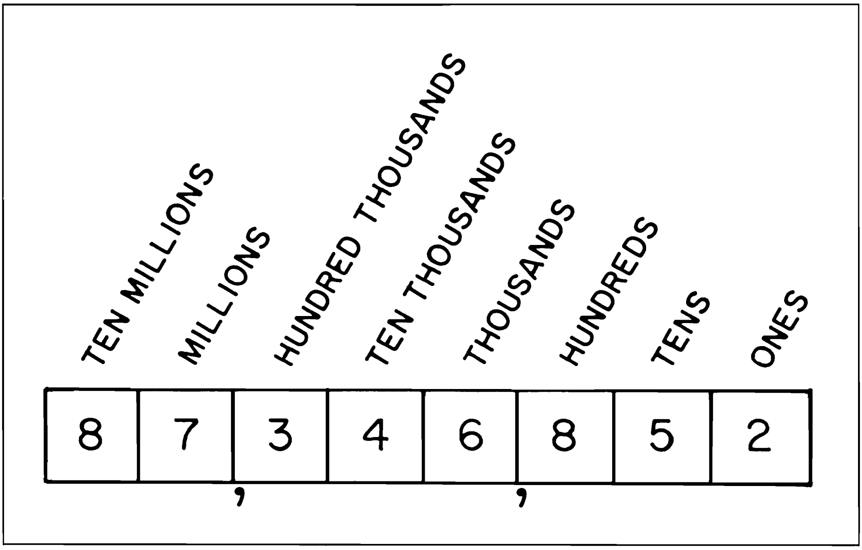


Figure 13-1

Each place value can also be expressed by taking ten to the power of that place number:

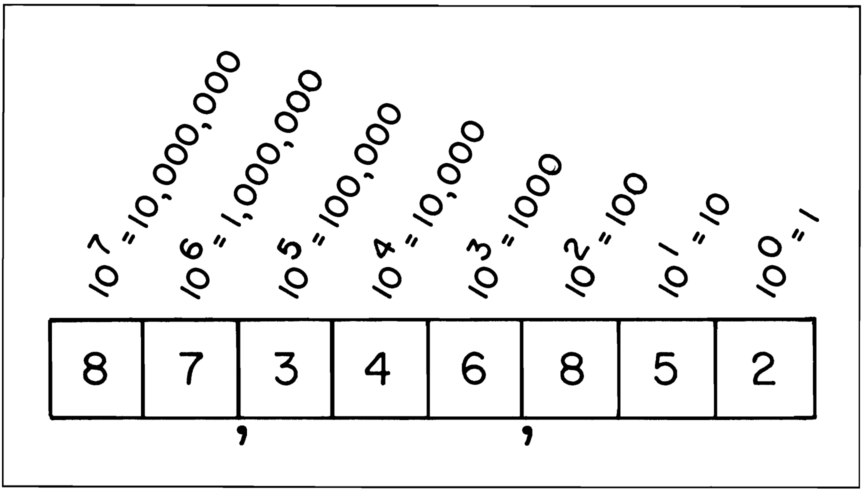


Figure 13-2

Notice that the low place values start on the right, and get larger as we move to the left.

Now let's do something you don't think about much. Let's take a decimal number and expand it according to the place values for each digit.

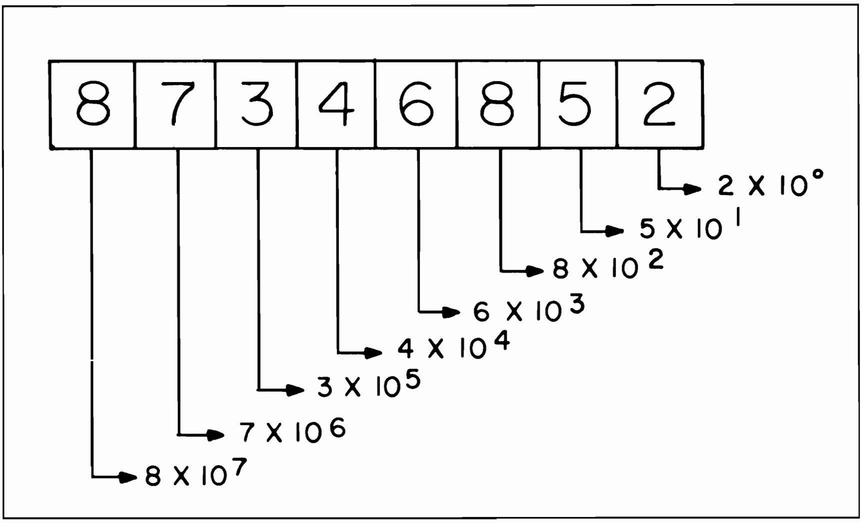


Figure 13-3

You probably looked at this exercise and thought "But that's so obvious!"

Yes, it does seem pretty silly. But only because we deal with decimal numbers day in and day out. The values of the various places in a decimal number are second nature to us.

Now let's switch to binary numbers and do the same thing.

CONSTRUCTING BINARY NUMBERS

The binary number system is based on the number 2. (That's why you will also hear it referred to as Base 2.) There are only two digits to use in binary numbers — zero and one. The place values for binary numbers are structured exactly the same as they are for decimal numbers, except the values are built around powers of 2 instead of powers of 10:

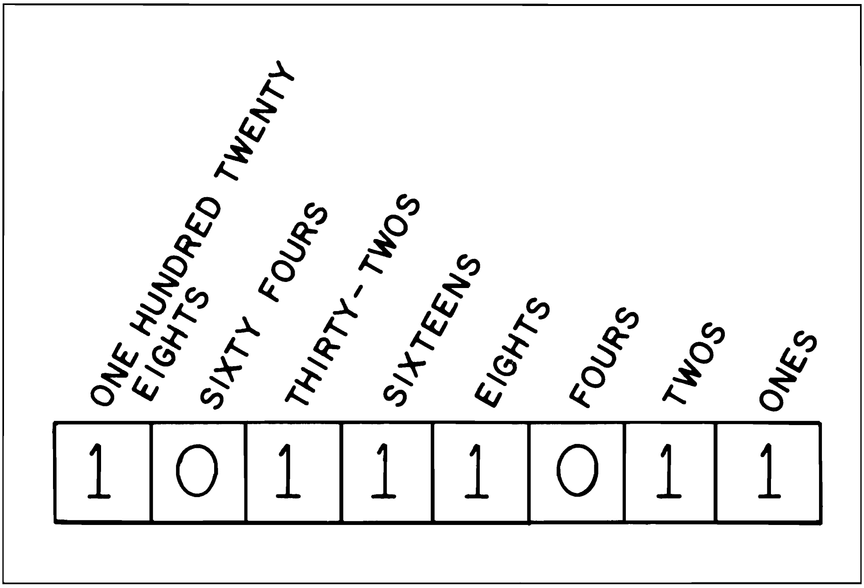


Figure 13-4

Binary place values can also be expressed as powers of 2:

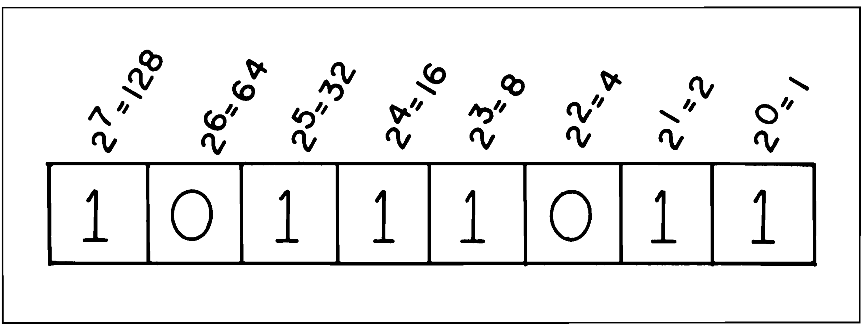


Figure 13-5

When we expand a binary number to find out what its equivalent is in decimal numbers, the chart of place values that we constructed comes in very handy. Now don't misunder-

stand — it's not that we need to convert these numbers from binary to decimal to have them MEAN something. The binary number 1101101 does have a definite meaning, and the computer can use it in that form. It's just that since we are so used to working with decimal numbers, we have trouble deciding if a binary value such as 1011 would be a better IQ or a shoe size. (By the way, the decimal equivalent of 1011 is 11, which would definitely be better as a shoe size.)

These examples of expanding binary numbers to find their decimal equivalents are illustrating the process we follow to convert Sprite 1s and zeroes from the Sprite grid into three 8-digit binary numbers, and then into their decimal equivalents. (As you may have noticed in Chapter 2, I recommend that you use the short Binary Conversion program included in your *Commodore 64 User's Guide* at the end of Chapter 6. You will want to do a few of these by hand, so you understand how they work, but converting all the binary numbers for 30 or 40 Sprites will kill an entire afternoon, if you don't let the computer do some of the work for you.)

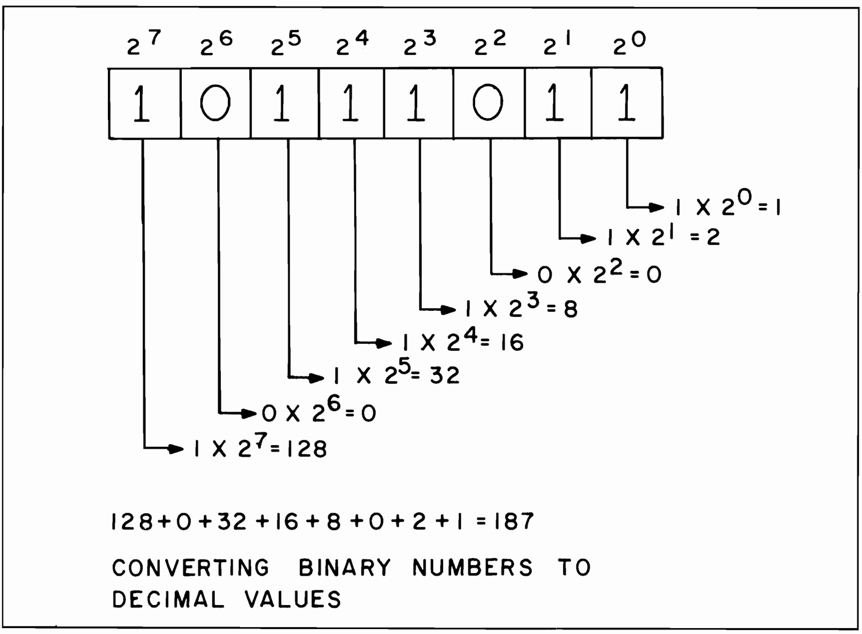


Figure 13-6

UNDERSTANDING BITS, BYTES, AND REGISTERS

Most of the time, when we want to place a value into the computer's memory, we can specify the decimal value:

```
10 LET X=45
20 POKE V+39,3
```

In both of these cases, we don't have to worry about the binary value of the numbers. The computer will do all the converting for us.

But working with Sprite registers is a slightly different matter. In this case, we are also specifying a decimal value to the computer, but we are also very concerned with how the binary equivalent of that decimal value will look. Before we do an example, let's look at how a register is constructed.

A register is a location in memory that has a special function. Depending on what values you store in that register, the computer will do different things. All the registers used with Sprite graphics are one byte long. The easiest way to envision a byte is that it is made up of eight bits, or tiny memory areas which can hold a binary 1 or a binary zero.

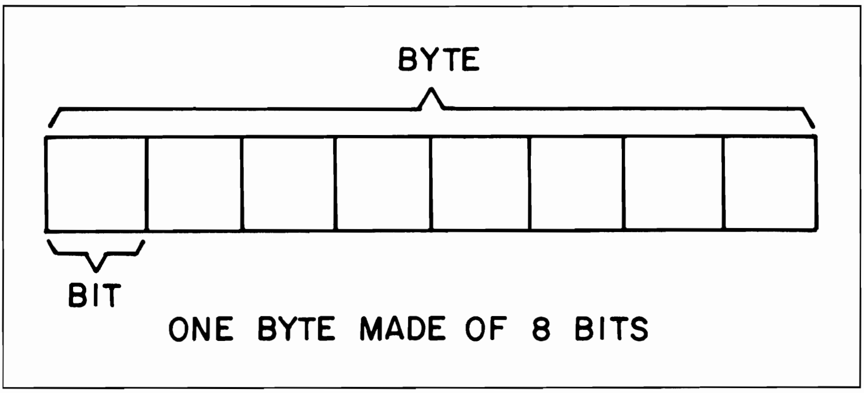


Figure 13-7

Normally, it takes one byte of memory to encode a single letter or character, such as the letter "T" or the character "&." Each letter and character has its own unique binary code in the memory. It takes a unique combination of eight binary 1s or zeroes to encode each letter of the alphabet or each character on the keyboard.

A byte in memory can also hold a number, instead of a letter of the alphabet. Understanding how binary numbers are coded in a register is the key to understanding Sprites.

A number stored in a register is nothing more than an 8-digit binary number. The place values are identical to those we saw earlier for binary numbers.

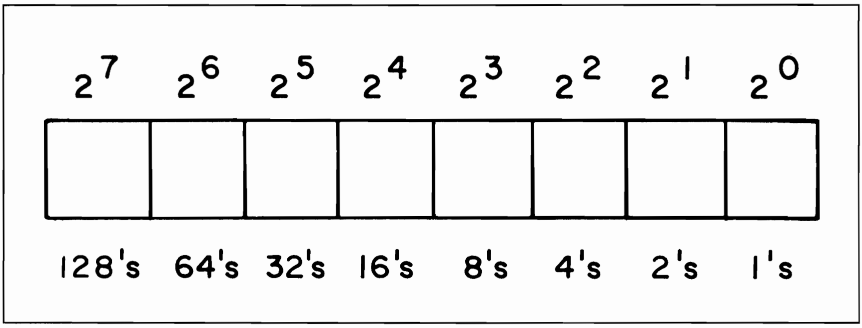


Figure 13-8

In addition to representing a place value, each bit in the register also represents one Sprite. Now you can see why only eight Sprites can be maintained by the video chip. Sprite graphics were designed to work with the fact that each register contains eight bits — one for each Sprite.

In addition, you can now understand why the Sprites are numbered from 0 through 7, not 1 through 8. Each Sprite is numbered after the exponential value for its bit in the register. Thus, the place value for the bit representing Sprite #0 is 2 to the zero power, or 1. The place value for Sprite #3 is 2 to the third power, or 8.

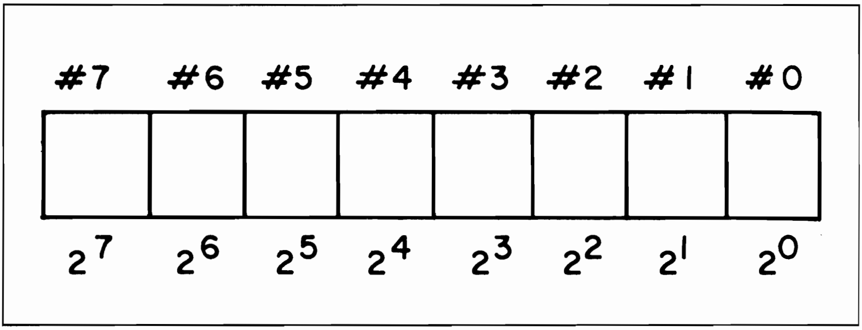


Figure 13-9

POKING VALUES INTO REGISTERS

Much of programming with Sprite graphics involves poking values into registers. For example, if you wish to enable Sprite #0 through #5, you must poke a value into the Sprite Enable Register, V + 16. Your goal is to place a 1 in the bits for Sprites #0 through 5, within the register located at location V + 16 within the video chip. You want the register to look like this:

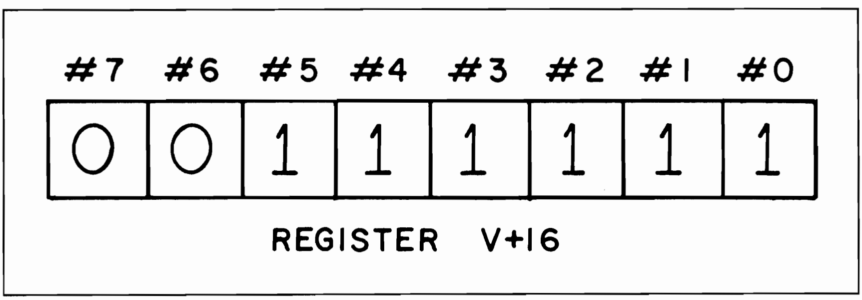


Figure 13-10

This job would be pretty simple if you had at your disposal a BASIC statement such as:

```
10 POKE V+16,BINARY 00111111
```

Unfortunately, there is no such animal. You are limited to poking a decimal number value into the register. So you must convert the binary number 00111111 to its decimal equivalent, and poke that decimal equivalent into the register. There are two ways to do this.

You can convert the binary number you need into a decimal value:

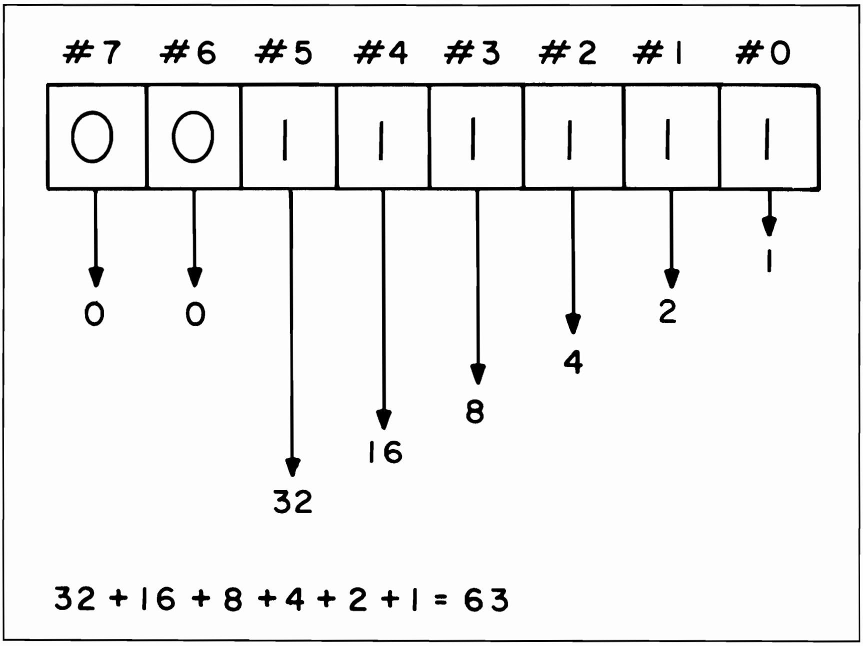


Figure 13-11

Then your POKE statement should read:

```
10 POKE V+16,63
```

Or, you can be lazy and let the computer do more of the work for you. Instead of adding all the binary place values yourself,

let the computer do it:

```
10 POKE V+16,1+2+4+8+16+32
```

(However, I must point out that this option executes less efficiently than doing the addition yourself.)

This same process works no matter which Sprites you wish to include, or which register you are working with. If you want to "turn on" a Sprite's bit in a register, that Sprite's place value must be added into the value you poke into the register.

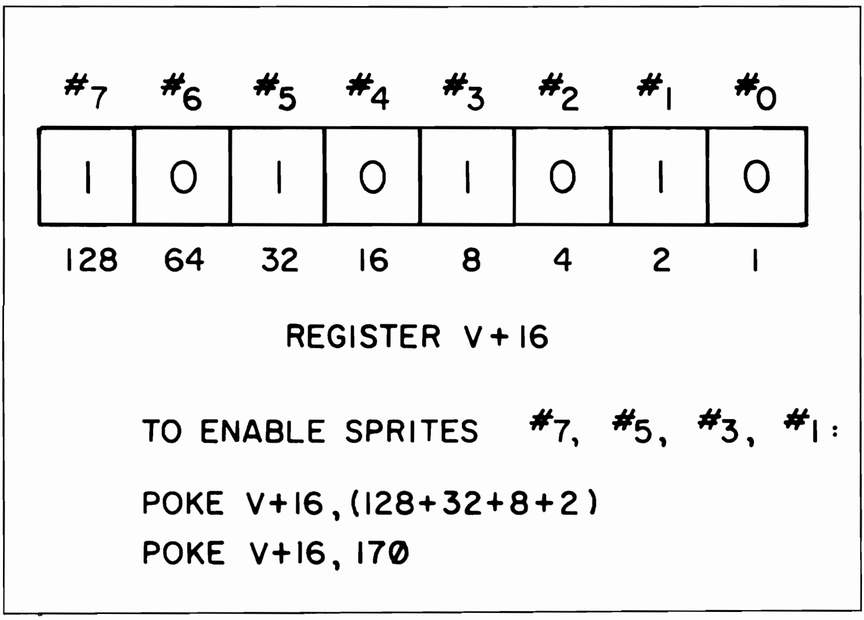


Figure 13-12

TURNING ON SELECTED BITS IN A REGISTER

As discussed in several of the earlier chapters, sometimes you need to be able to turn on or off just one bit within a register without affecting the other bits. To accomplish this, you will

need to use a Boolean operation involving the AND and OR functions.

This is the general format of the statement for turning ON a selected bit within a register:

```
10 POKE (V+?),PEEK(V+?) OR (2↑SN)
```

In this general statement, (V + ?) is the register location. It will be the same in both cases. SN stands for the Sprite number from #0 through 7.

The value to the right of the comma will be poked into the register. We'll follow in detail how that value is figured.

PEEK(V + ?) means to "look" into register V + ? to see the value stored there. Then an OR operation is performed on that value, using a second value of (2!SN). As you recall, (2!SN) is the place value for a Sprite.

This is probably still pretty confusing, so let's choose a more concrete example.

Let's assume that the Sprite Enable Register (V + 21) originally looked like this:

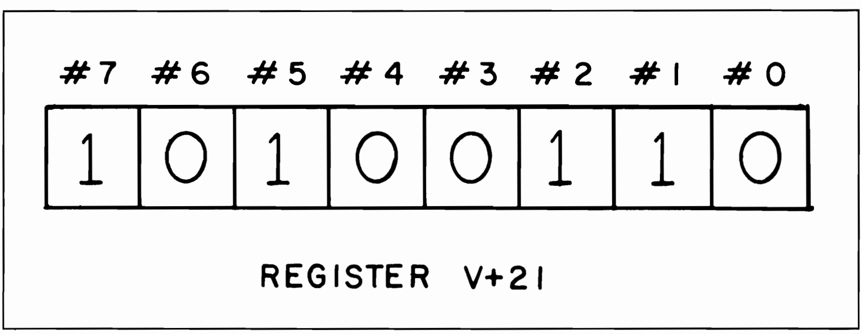


Figure 13-13

We are interested in turning on only the bit in the Sprite Enable Register (V + 21) for Sprite #4. The place value of the bit for Sprite #4 is 2 to the fourth power, or 16. This is the statement we will need to use:

```
10 POKE (V+21),PEEK(V+21) OR (2↑4)
```

By using the OR, we are instructing the computer to perform a Boolean OR on the binary value found in the register, and the binary value of 2!4, or 16. A Boolean OR compares the corresponding bits in the two values and changes them according to this table of rules:

0 or 0 = 0
0 or 1 = 1
1 or 0 = 1
1 or 1 = 1

In our case, here are the results:

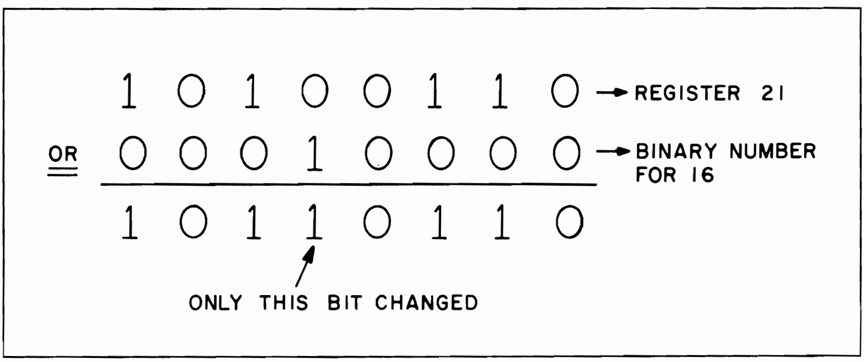


Figure 13-14

The resulting value in the register is different only in the bit for Sprite #4. It used to be a zero, and now it is a one. No other bit was changed.

But what would have happened if the bit for Sprite #4 had already been turned on?

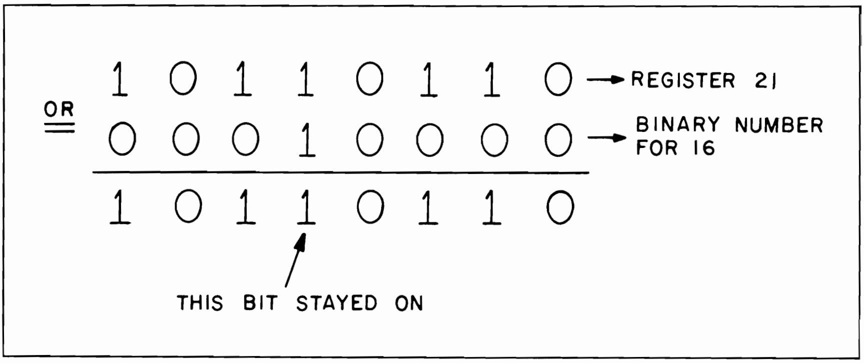


Figure 13-15

As you can see, the bit for Sprite #4 was left on, which is what we wanted. The other bits were not changed.

TURNING OFF SELECTED BITS IN A REGISTER

Turning OFF selected bits within a register, without changing any of the other bits, is accomplished in a similar fashion.

The general format of the statement to turn off a selected bit is:

```
10 POKE (V+?),PEEK(V+?) AND (255-2↑SN)
```

where (V + ?) is the register location, which will be the same in both cases, and SN is the Sprite number from #0 through 7.

Again, the computer PEEKs at the binary value stored within the register, then performs a Boolean AND operation on that value and (255-2!SN).

A Boolean AND compares the corresponding bits in the binary values and changes them according to the rules in this table:

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

If we wish to turn OFF the bit for Sprite #4 in the Sprite Enable Register (V+21), we would use this statement:

```
10 POKE (V+21),PEEK(V+21) AND (255-2↑4)
```

The value of the arithmetic expression (255-2!4) is 239. The binary equivalent of this number is 11101111.

This diagram shows the original contents of the Sprite Enable Register, and the results when the AND operation is performed. Notice that the bit for Sprite #4 was "on" when we started.

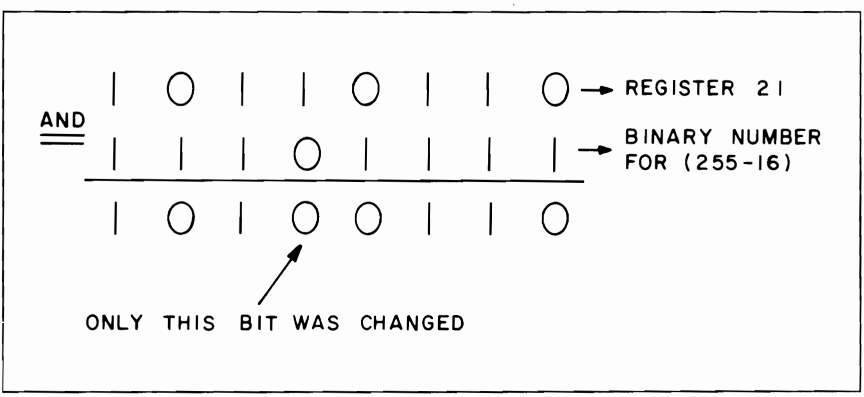


Figure 13-16

Even if the bit for Sprite #4 was already "off" before using the Boolean AND operation, the results are the same:

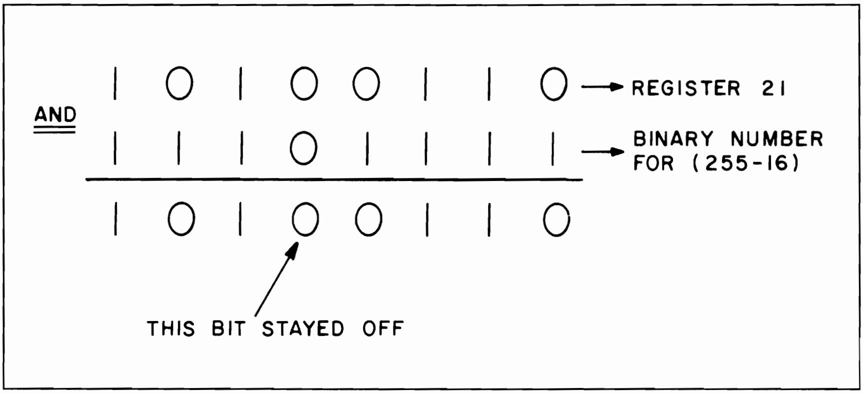


Figure 13-17

As you may have guessed, these statements to selectively turn on and off bits within a register can also be used to change more than one bit at a time. All that is involved is changing the (2!SN) value to the sum of the place values of the Sprites involved. For example, to turn ON Sprites #1 and #2:

```
10 POKE (V+21),PEEK(V+21) OR 6
```

The place values for Sprites #1 and #2 are 2 and 4, respectively. The sum of 2 and 4 is 6, so that is the value used.

To turn OFF the bits for Sprites #5 and #7, the statement would look like this:

```
10 POKE (V+21),PEEK(V+21) AND (255-160)
```

The place values for Sprites #5 and #7 are 32 and 128. Their sum is 160, which then must be subtracted from 255.

CHAPTER 14

QUESTIONS AND ANSWERS ABOUT PROGRAMMING WITH SPRITES

A Spritely Mess?

Q. The DATA statements for my Sprite look correct, meaning that I have 63 data values, and they're in the correct order. Yet, when I execute my program, the Sprite looks like a terrible mess on the screen. I've looked for typos in my DATA values, but nothing is wrong. What could be my problem?

A. Several things come to mind. First, are you certain you are poking the data values into the correct memory locations? If you designate in the Sprite Pointer address 2040 that Sprite #0 will have its data values in memory area 192, then they'd better *be* in (192*64) +0 through (192*64) +62 plus one empty byte at (192*64) +63 as a place holder. A typo in your program statements to read these data values and POKE them into memory could cause the problem.

Is the Sprite multi-colored and you have neglected to designate it as such? If a Sprite is multi-colored, you must turn on its bit in the Sprite Multicolor Register (V+28). Sometimes this error will result in a Sprite that looks faintly recognizable (if you used mostly background color and multi-

color 2 in designing the Sprite) and sometimes will just be a random mess (if you used mostly multi-color 1 and the Sprite color for designing the Sprite).

If you have more than one Sprite in your program, and it's the second Sprite that looks funny, maybe you were short a data value or two for the first Sprite. Let me give you an example. If Sprite #0 had only 61 data values (let's say you forgot to type in the final 255,255), then when the program reads in the 63 data values for Sprite #0, it will read the 61 values given for Sprite #0, then go on to read in the first two numbers for Sprite #1, to make a total of 63. Now, if the first two values for Sprite #1 happen to be the same as the two values missing from the end of Sprite #0 (and this happens more often than you would imagine), then Sprite #0 will look normal on the screen. However, Sprite #1 will be two values off. This mistake could easily escape your attention if you had more DATA values in your program than you were planning to use. Otherwise, of course you would get an "OUT OF DATA" error as the program tried to read in the final two data values for Sprite #1 and found none.

A third possibility could be that you have a stray comma at the end of a DATA statement. The Commodore 64 assumes that data elements are separated by commas. If you have a DATA line which ends with a comma, the program will try to read an additional blank data value after that comma, throwing all your Sprite values off by one place. If this occurred halfway through your Sprite data values, the top part of the Sprite would look normal and the bottom would look funny. If the extra comma occurred in the first few data statements, the whole Sprite would probably look goofy.

Handy Tricks with Binary Numbers

Q. I'm not very proficient with binary numbers yet. Could you give me a few "slick tricks" to make my life easier? For instance, if I want to figure out the total decimal place value for Sprites #0 through 6, is there an easier way to do it than just adding all the individual place values?

A. A useful rule to remember is that the total of a series of place values is always one less than the next place value. For example, to figure the decimal equivalent of the binary number 111111: the highest place value in this number is 2^5 or 32. The next higher place value would be 2^6 or 64. So the decimal equivalent of 111111 is $(2^6 - 1)$ or 63. To prove to yourself that this works, add up the place values. $1+2+4+8+16+32=63$. Remember, though, that this rule only works if the binary number consists of all ones.

101 Sprites?

Q. How do I get more than eight Sprites on the screen at once?
A. Use Raster Interrupt techniques. Unfortunately, you're on your own here, because those techniques are beyond the scope of this book.

Safe Memory

Q. Are there any other "safe" memory areas for storing Sprite data, besides the 64 areas beginning with area 192?
A. Commodore suggests areas 13, 14, and 15 (the cassette buffer) if you will be defining three Sprites or less.

The Phantom Sprite Strikes Again

Q. I've enabled a Sprite, but I can't see it on the screen. I know it isn't the same color as the background, because I checked. What's wrong?
A. You've probably assigned it X and Y coordinates that are off the visible part of the screen. Try changing the X and Y values for the Sprite in direct mode, to see if you can make it appear. Then don't forget to change the values in your program, too.

Detecting Fake Collisions

Q. Occasionally my program will detect a collision when I know there hasn't been one. What could be causing this?
A. At least three things could be happening here. All of these are equally likely to happen whether you are looking at the Sprite to Sprite collision register (V+30) or the Sprite to

background collision register (V+31).

1) Bits in the collision register may have been turned "on" when your program started executing. Therefore, the first time you PEEKed at the register contents, it indicated a collision had taken place. The solution to this is to have a "dummy" PEEK statement early in the program, solely for the purpose of clearing the register. Remember that the register is automatically set to zero when you PEEK at the contents.

2) The collisions may actually be occurring, but are happening off the visible portion of the screen, so you are not aware of them.

3) Your Sprites are colliding with "sparkles" (little blips of light) on the screen, giving a false collision. This is the hardest situation to deal with, since it's so unpredictable. My own equipment does this so seldom that I have very little experience in dealing with it. However, I've been told that one remedy is to PEEK at the register twice in a row, and to only believe the results of the second PEEK. This way, you're betting that a sparkle won't cause you trouble twice in a row. (It's sort of like depending on lightning never striking twice in the same place. Nobody gets any guarantees.)

Files of Just Sprite Data Values

Q. I made up a large set of Sprite data values for the letters of the alphabet and all the numerals. I want to use these Sprites for several programs. It seems rather absurd to save all these DATA values over and over, but with different program statements in front of them. Isn't there a better way to do this?

A. Yes, there is. You can take your cue from the techniques used by several of the software packages you can buy for creating Sprite data values. They create a file of Sprite values, separate from any program statements, and instruct you to save this file on a disk or tape. Then when you want to use these Sprite data values in a program, you load the file into memory using the special parameters of the LOAD statement which tell the computer to load the file back into the same memory locations from which it was saved. (This

statement is LOAD "file name," device number,1 where "1" tells the computer to load the values back into the same memory locations they were saved from. This accomplishes the same thing as reading in all the DATA statement values and poking them into memory. The only thing you must remember is which Sprite data you put where in memory. If you have a large number of Sprites, this is much more efficient than coding that huge collection of DATA values in each program.

Alternate Bank Deposit

Q. I've noticed that all your examples place the Sprite DATA values in memory locations within memory bank zero. Can I use the other banks? How would I do this?

A. I'll refer you to an expert for this answer. Tim Villanueva has an excellent article in the Summer 1983 *Commodore Power/Play* magazine on just this subject. It appears on pages 64-66.

Backward or Upside Down Sprites

Q. I designed several Sprites for a game program, and now I find I could really use another version of one of the Sprites, except I'd like it to face the opposite direction, or be upside down. Is there any way to do this without having to start over again with the Sprite grid and code the new views from scratch?

A. The three following sample routines will solve all your problems. First, here is the usual way we would read and poke the DATA values for a Sprite. The Sprite I have chosen is facing left after we run this program.

```
10 REM NORMAL SPRITE PROGRAM
20 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281,1:POKE 53280,2
50 POKE 2040,192
60 FOR M=0 TO 62
80 READ A
90 POKE 192*64+M,A
```

```
110 NEXT M
115 POKE V+39,0
118 POKE V+23,1:POKE V+29,1
120 POKE V+0,150 :POKE V+1,150
130 POKE V+21,1
140 END
10000 REM MEAN WOLF SPRITE
10010 DATA 0,0,0,0,0,0,0
10020 DATA 0,48,1,192,112
10030 DATA 0,240,240,0,121
10040 DATA 224,0,61,192,0
10050 DATA 31,192,0,15,224
10060 DATA 0,63,224,255,255
10070 DATA 240,255,193,248,255
10080 DATA 255,252,170,255,254
10090 DATA 0,255,254,0,255
10100 DATA 254,0,255,254,170
10110 DATA 255,252,255,255,248
10120 DATA 1,255,240,1,255,240
```

Now, if we want to take these same 63 data values and make an upside down wolf Sprite, we make a few changes in the order in which we read in the data values. Instead of reading in the values from top to bottom, we need to read them in from bottom to top. But we also need to reverse the order of the three Sprite values in each row. Let me show you how statements 60-110 work.

```
10 REM UPSIDE DOWN SPRITE ROUTINE
20 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281,1:POKE 53280,2
50 POKE 2040,192
60 FOR M=62 TO 0 STEP -3
70 FOR N=2 TO 0 STEP -1
80 READ A
90 POKE 192*64+(M-N),A
100 NEXT N
110 NEXT M
115 POKE V+39,0
118 POKE V+23,1:POKE V+29,1
```

```
120 POKE V+0,150 :POKE V+1,150
130 POKE V+21,1
10000 REM MEAN WOLF SPRITE
10010 DATA 0,0,0,0,0,0,0
10020 DATA 0,48,1,192,112
10030 DATA 0,240,240,0,121
10040 DATA 224,0,61,192,0
10050 DATA 31,192,0,15,224
10060 DATA 0,63,224,255,255
10070 DATA 240,255,193,248,255
10080 DATA 255,252,170,255,254
10090 DATA 0,255,254,0,255
10100 DATA 254,0,255,254,170
10110 DATA 255,252,255,255,248
10120 DATA 1,255,240,1,255,240
```

For a normal Sprite, we poke the values into memory areas in the order they are read. The memory addresses range from the memory area plus 0 to the memory area plus 62.

Line 60 says we will start at +62 and work our way back to +0 by threes. In other words, we need to do something special with each group of three values. In statement 70, what we are doing with each group of three values (representing one row of the Sprite grid) is to place the data values in normal order within each group of three, instead of going backwards as we are doing with all the data values as a whole. This probably seems very confusing to you, but trust me, it works. The only way you will really understand it is to get out a pencil and a sheet of paper and try it for yourself.

Now suppose your want a Sprite that is backward, meaning that instead of my wolf Sprite facing left, I want one that looks just the same, but is facing right. This is a bit more complicated, but bear with me and I'll try to explain how it works.

```
10 REM BACKWARD SPRITE ROUTINE
20 V=53248
30 PRINT CHR$(147):POKE V+21,0
```

```
40 POKE 53281,1:POKE 53280,2
50 POKE 2040,192
60 FOR M=0 TO 62 STEP 3
70 FOR N=2 TO 0 STEP -1
80 READ A
85 GOSUB 500
90 POKE 192*64+(M+N),A2
100 NEXT N
110 NEXT M
115 POKE V+39,0
118 POKE V+23,1:POKE V+29,1
120 POKE V+0,150 :POKE V+1,150
130 POKE V+21,1
140 END
500 REM SUBROUTINE TO REVERSE BINARY NO.
505 IF A=255 OR A=0 THEN A2=A:GOTO 610
510 A2=0
520 FOR P=7 TO 0 STEP -1
530 D=INT(A/2↑P)
540 IF D=0 THEN GOTO 600
550 A2=A2+2↑(7-P)
560 A=A-(D*2↑P)
600 NEXT P
610 RETURN
10000 REM MEAN WOLF SPRITE
10010 DATA 0,0,0,0,0,0,0
10020 DATA 0,48,1,192,112
10030 DATA 0,240,240,0,121
10040 DATA 224,0,61,192,0
10050 DATA 31,192,0,15,224
10060 DATA 0,63,224,255,255
10070 DATA 240,255,193,248,255
10080 DATA 255,252,170,255,254
10090 DATA 0,255,254,0,255
10100 DATA 254,0,255,254,170
10110 DATA 255,252,255,255,248
10120 DATA 1,255,240,1,255,240
```

This time we are going to read the Sprite data into memory in top to bottom order, as we normally would, but we will change the order for each set of three data values (representing one row in the Sprite grid). The data value that

would have described the far right side of the Sprite needs to be on the far left side now, and vice versa for the data value that normally would have been describing the left side of the Sprite shape. The middle number will stay where it has always been. Now comes the tricky part. It isn't enough to reverse the numbers alone, but the actual bits for each number must also be reversed. If the original binary number was 1011100, the reversed number must look like 0011101 (just like holding the number up to a mirror). The subroutine that starts at line 500 does this. It figures out the binary place digit (either zero or one), beginning at the far left side, then uses that digit to build a new reversed binary number, starting with the binary places on the far right side. In this routine, A is the original decimal DATA number, A2 is the decimal value of the final reversed number, P stands for the power of two being used, and D is the binary digit. I will forewarn you that this routine takes time to execute for every one of the 63 data values in your Sprite. However, I did speed it up quite a bit by realizing that two commonly found data values, 255 and zero, will not change at all when put through this algorithm. So I added line 505 to bypass all the seven "loops" in the subroutine if A is either 255 or zero.

The backward and upside down Sprite routine is relatively easy to understand, once you have mastered the first two. It reads the data values and pokes them into memory in exactly reverse order. The data value it pokes is the "reverse" value, the same as we needed for the backward Sprite routine.

```
10 REM BACKWARD AND UPSIDE DOWN SPRITE
20 V=53248
30 PRINT CHR$(147):POKE V+21,0
40 POKE 53281,1:POKE 53280,2
50 POKE 2040,192
60 FOR M=62 TO 0 STEP -1
80 READ A
85 GOSUB 500
90 POKE 192*64+M,A2
110 NEXT M
```

```
115 POKE V+39,0
118 POKE V+23,1:POKE V+29,1
120 POKE V+0,150 :POKE V+1,150
130 POKE V+21,1
140 END
500 REM SUBROUTINE TO REVERSE BINARY NO.
505 IF A=255 OR A=0 THEN A2=A:GOTO 610
510 A2=0
520 FOR P=7 TO 0 STEP -1
530 D=INT(A/2↑P)
540 IF D=0 THEN GOTO 600
550 A2=A2+2↑(7-P)
560 A=A-(D*2↑P)
600 NEXT P
610 RETURN
10000 REM MEAN WOLF SPRITE
10010 DATA 0,0,0,0,0,0,0
10020 DATA 0,48,1,192,112
10030 DATA 0,240,240,0,121
10040 DATA 224,0,61,192,0
10050 DATA 31,192,0,15,224
10060 DATA 0,63,224,255,255
10070 DATA 240,255,193,248,255
10080 DATA 255,252,170,255,254
10090 DATA 0,255,254,0,255
10100 DATA 254,0,255,254,170
10110 DATA 255,252,255,255,248
10120 DATA 1,255,240,1,255,240
```

If you need backward or upside down Sprites for use within a program, be sure to use these techniques while you read and poke the DATA values into memory. It is much too time-consuming to use these routines during the course of moving your Sprites within the program's action.

If you use these routines with multi-colored Sprites, they will work, but the colors for multi-color 2 and the background will be switched, due to the way they are normally coded in the Sprite grid.

Even though these routines are coded in "uncrunched" form for readability, be sure to crunch them in your programs. The efficiency will be greatly improved.

CHAPTER 15

SPRITE REGISTER SUMMARY

SPRITE REGISTERS

V = 53248	SPRITE #7	SPRITE #6	SPRITE #5	SPRITE #4	SPRITE #3	SPRITE #2	SPRITE #1	SPRITE #0
DECIMAL PLACE VALUE EQUIVALENTS	128	64	32	16	8	4	2	1
SPRITE ENABLE REGISTER	V+21,128	V+21,64	V+21,32	V+21,16	V+21,8	V+21,4	V+21,2	V+21,1
SPRITE POINTER	2047	2046	2045	2044	2043	2042	2041	2040
SUGGESTED MEMORY AREA*	199	198	197	196	195	194	193	192
SPRITE COLOR	V+46,C	V+45,C	V+44,C	V+43,C	V+42,C	V+41,C	V+40,C	V+39,C
X-POSITION	V+14,X	V+12,X	V+10,X	V+8,X	V+6,X	V+4,X	V+2,X	V+0,X
Y-POSITION	V+15,Y	V+13,Y	V+11,Y	V+9,Y	V+7,Y	V+5,Y	V+3,Y	V+1,Y
MOST SIGNIFICANT BIT (FOR RIGHT X)	V+16,128	V+16,64	V+16,32	V+16,16	V+16,8	V+16,4	V+16,2	V+16,1
HORIZONTAL EXPANSION	V+29,128	V+29,64	V+29,32	V+29,16	V+29,8	V+29,4	V+29,2	V+29,1
VERTICAL EXPANSION	V+23,128	V+23,64	V+23,32	V+23,16	V+23,8	V+23,4	V+23,2	V+23,1
MULTI-COLOR MODE	V+28,128	V+28,64	V+28,32	V+28,16	V+28,8	V+28,4	V+28,2	V+28,1
SPRITE/BACKGROUND PRIORITY (SPRITE PASS BEHIND)	V+27,128	V+27,64	V+27,32	V+27,16	V+27,8	V+27,4	V+27,2	V+27,1

* MEMORY AREA 192 means (192*64)+0 through (192*64)+62

MULTI-COLOR 1	POKE V+37,C
MULTI-COLOR 2	POKE V+38,C

These two colors will apply to all Sprites designated as multi-colored.

SCREEN BACKGROUND	POKE 53281,C
SCREEN BORDER	POKE 53280,C

Reprinted courtesy of Commodore Business Machines, Inc.

SPRITE PRIORITIES

Lower numbered Sprites have priority over higher numbered Sprites. If Sprites #2 and #6 cross paths, Sprite #2 will appear to pass in front of Sprite #6.

SPRITE TO SPRITE COLLISION

```
10 PEEK(V+30) AND X =X THEN [TAKE SOME ACTION]
```

total of decimal place value equivalents for Sprites you wish to check

SPRITE TO BACKGROUND FIGURE COLLISION

```
20 PEEK(V+31) AND X =X THEN [TAKE SOME ACTION]
```

total of decimal place value equivalents for Sprites you wish to check

TO TURN *ON* SELECTED BITS IN A REGISTER

```
30 POKE V+?,PEEK(V+?) OR (2↑SN)
```

register number

Sprite number (0 through 7)

TO TURN *OFF* SELECTED BITS IN A REGISTER

```
40 POKE V+?,PEEK(V+?) AND (255-2↑SN)
```

register number

Sprite number (0 through 7)

SPRITE SIZE

Normal: 24 dots wide by 21 dots high
Fully Expanded: 48 dots wide by 42 dots high

SPRITE REGISTER SUMMARY

CODING MULTI-COLOR SPRITES

00 Background Color
01 Multi-color 1
10 Sprite Color
11 Multi-color 2

CODING SINGLE-COLOR SPRITES

0 Background Color
1 Sprite Color

COLOR CODES

0	Black	8	Orange
1	White	9	Brown
2	Red	10	Light Red
3	Cyan (Light Blue)	11	Dark Gray
4	Purple	12	Medium Gray
5	Green	13	Light Green
6	Blue	14	Light Blue
7	Yellow	15	Light Gray

A SPRITE REGISTER And its Decimal Place Value Equivalents

#7	#6	#5	#4	#3	#2	#1	#0
$2^7=128$	$2^6=64$	$2^5=32$	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$

VALID X AND Y CORRDINATE VALUES*

Y can be 0 through 255
X can be 0 through 255, but if values are needed *past* 255, turn on the MSB (Most Significant Bit) for that Sprite, then start over with your X-coordinate at 0

*Some of these X and Y values will be outside the visible screen area

INDEX

PERSONAL COMPUTERS

SPRITE GRAPHICS FOR THE COMMODORE 64 is the definitive book which unlocks the astonishing visual capabilities of the Commodore 64. Designed for the beginning and intermediate computer user, SPRITE GRAPHICS FOR THE COMMODORE 64 leads the computerist through the steps that will—

- ★ Define Sprites of all imaginable shapes
- ★ Color Sprites with the hues of the rainbow
- ★ Animate Sprites and set them in motion
- ★ Construct games through Sprite collision detection
- ★ Answer every Sprite question—and much, much more

Written in clear, non-technical language, and complete with tables, graphs, and scores of ready-to-run programs, SPRITE GRAPHICS FOR THE COMMODORE 64 is essential to fully enjoying the Commodore 64 computer.


Bill McConnell

ABOUT THE AUTHOR

Sally Greenwood Larsen is well known in the world of computer book publishing. She is a pioneer in the teaching of computer programming to young people, and has published eight best-selling books on the subject. Ms. Larsen has been a teacher and a college computer department administrator, and is now a computer programmer for Deere & Company in Moline, Illinois, specializing in data base applications. She attended the University of Wisconsin at Madison, majoring in education with concentrations in mathematics and science.

Micro Text Publications, Inc.
Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632

0 21898 83813 1

Cover design by Hal Siegel

3
ISBN 0-13-838136-4